Monte Carlo Numerical Evaluation of a Definite Integral - Hit and Miss Method

Created using Maple 14.01

Jake Bobowski

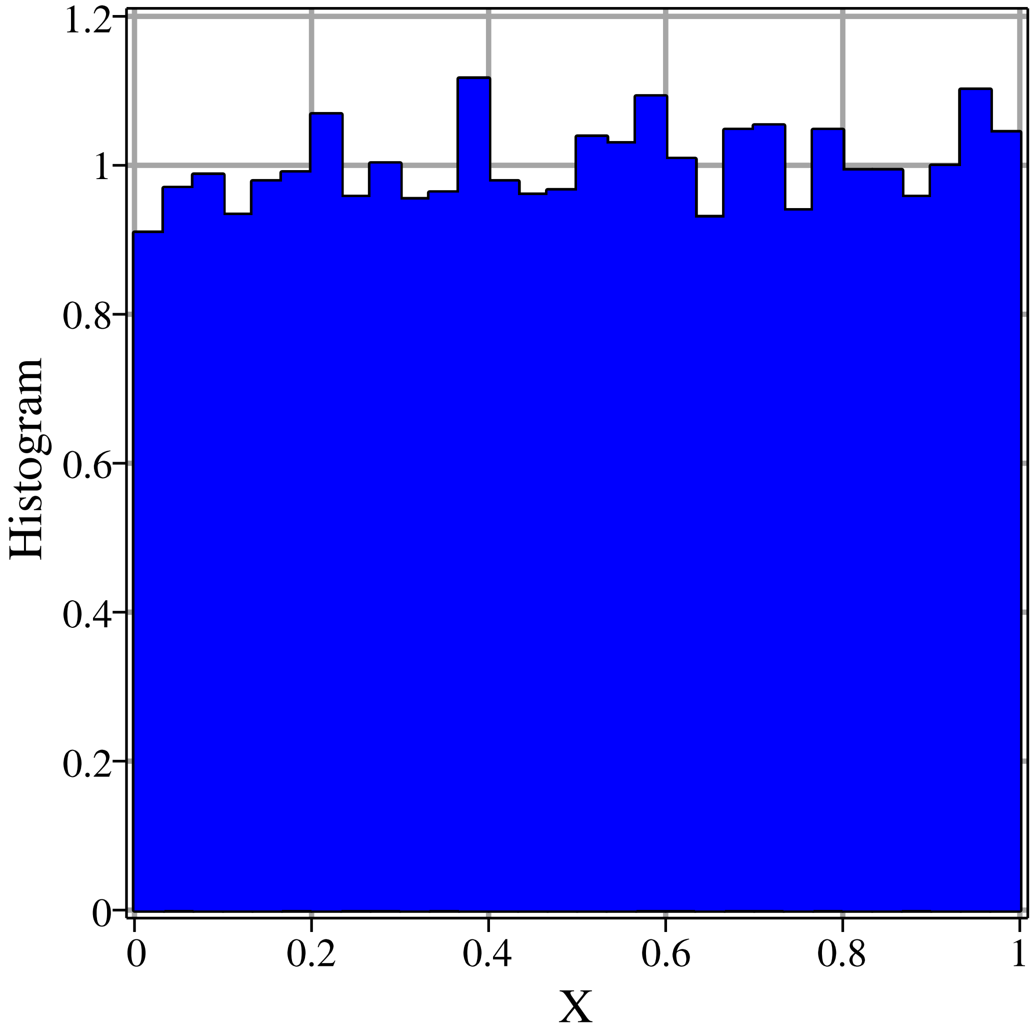
```
> restart;
  with(stats) :
   with(plots) :
   with(Statistics) :
  with(StringTools) :
   FormatTime("%m−%d−%Y, %H:%M");
                              "03-21-2013, 21:32"                        (1)
```

In all Monte Carlo simulations it is necessary to generate random or pseudo-random numbers. The following statement will generate a random number drawn from a uniform distribution between 0 and 1.

```
> X := x→stats[random, uniform[0, 1]](1) :
   X( );
                                0.3957188605                             (2)
```

To confirm that the numbers generated by $X(\,)$ are uniformly distributed on the interval [0,1], we can generate a long list of these numbers and plot a histogram.

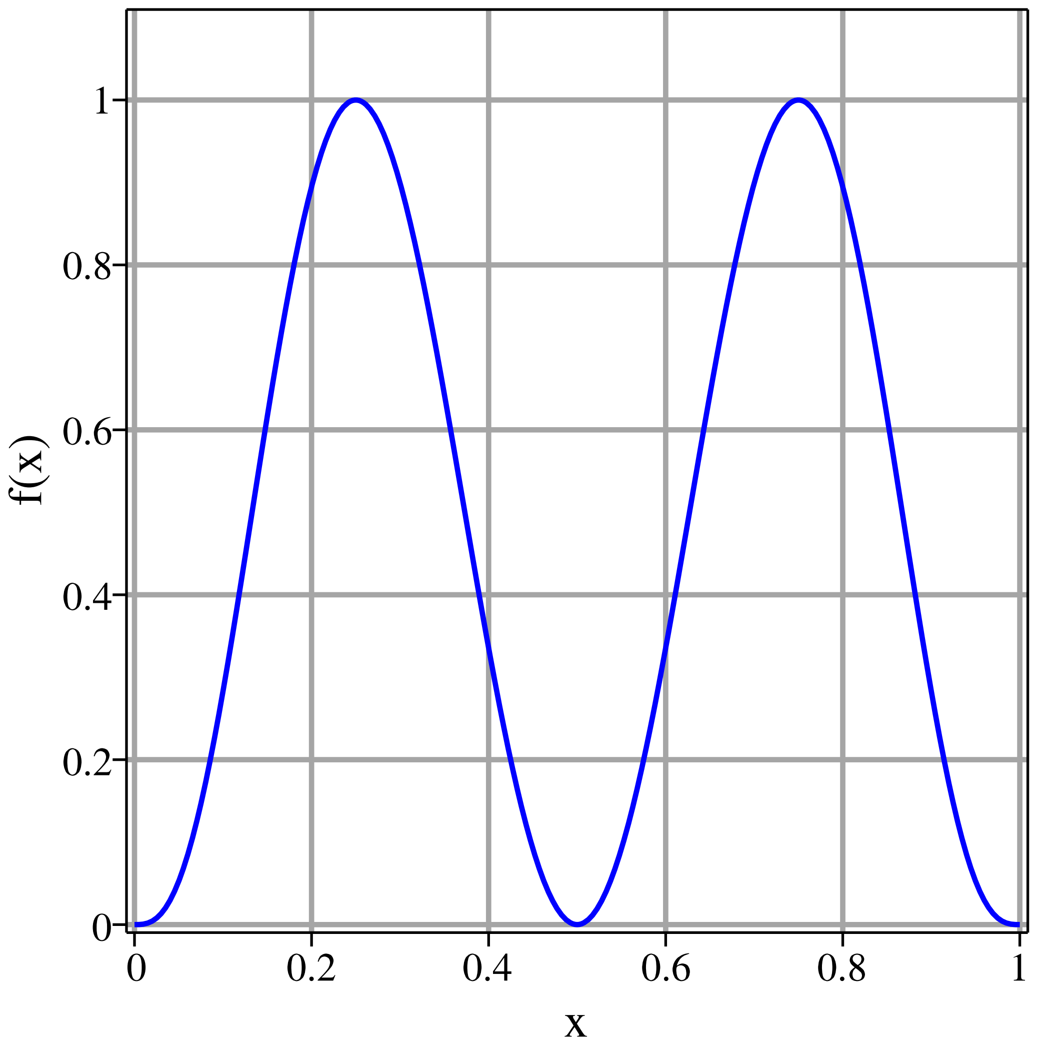
```
> XList := NULL :
   for i from 1 to 10e3 do:
   XList := XList, X( ) :
   end do:
   XList := [XList] :
   Histogram(XList, axes = boxed, view = [0 .. 1 , 0 .. 1.2], labels = [typeset("X"),
       typeset("Histogram") ], labeldirections = ["horizontal", "vertical"], symbol = circle,
       symbolsize = 20, thickness = 2, tickmarks = [8, 8], colour = blue , axesfont = [Times, 12] ,
       labelfont = [Times, 14], axis = [gridlines = [thickness = 2]]);
```



This tutorial will attempt to numerically evaluate an integral for which the exact solution is easily obtained. This approach has been taken purposely so that we can confirm that our numerical techniques are reliable. The function that we will integrate is a simple polynomial. Below the function is plotted on the interval x = 0..1 and the exact value of the integral is evaluated over the same interval.

> $fcn := \frac{1}{27}\left(-65536\cdot x^8 + 262144\cdot x^7 - 409600\cdot x^6 + 311296\cdot x^5 - 114688\cdot x^4 + 16384\cdot x^3\right)$:

```
plot(fcn, x = 0 ..1, axes = boxed, view = [0 .. 1, 0 .. 1.1 ], labels = [typeset("x"),
    typeset("f(x)") ], labeldirections = ["horizontal", "vertical"], symbol = circle, symbolsize
    = 20, thickness = 2, tickmarks = [8, 8 ], colour = blue , axesfont = [Times, 12 ] , labelfont
    = [Times, 14 ], axis = [gridlines = [thickness = 2 ]]);
IntExact := evalf( int(fcn, x = 0 ..1 ) );
```



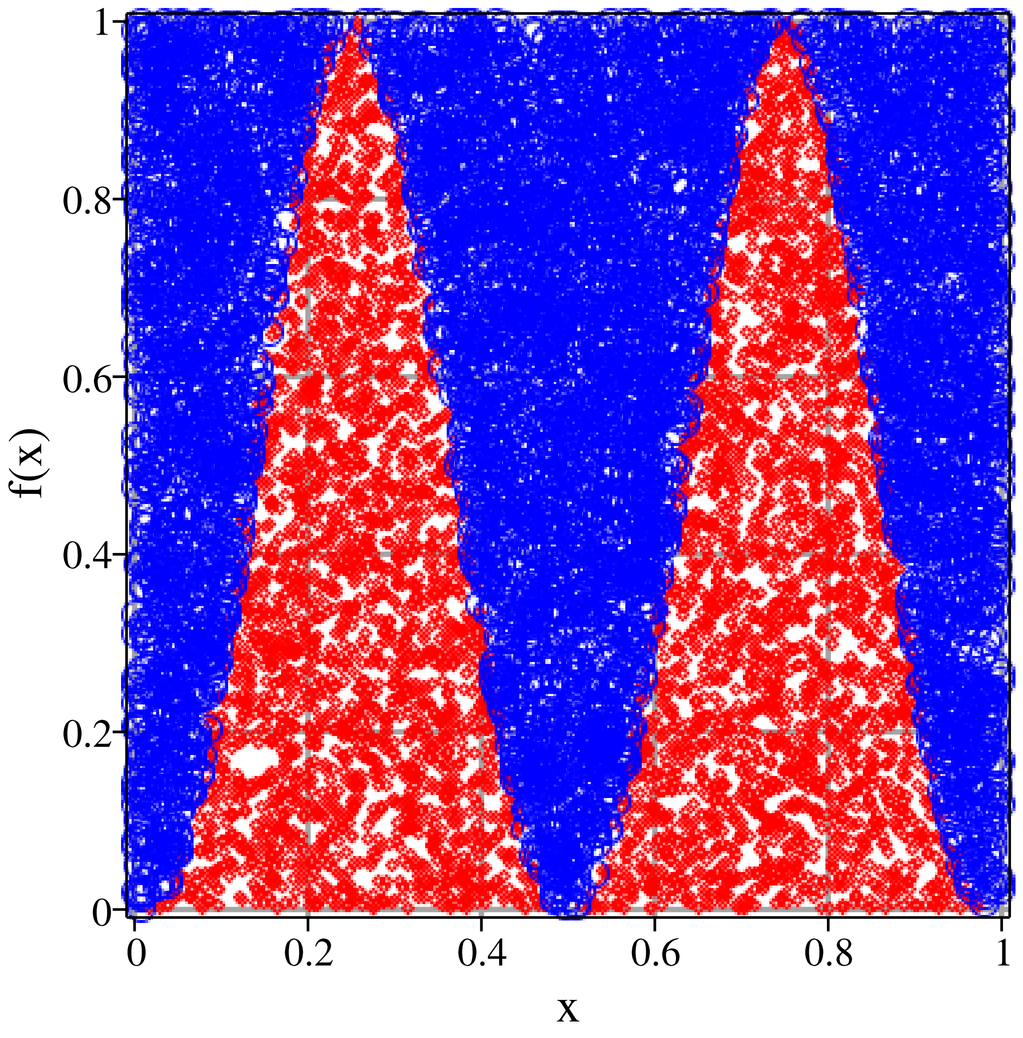
$IntExact := 0.4815990594$ **(3)**

The first method that we will implement is the *Hit & Miss* technique. In this method pairs of random numbers (x i, y i) will be generated. For the x-coordinate the x i values will be between 0 and 1 (the integration interval). Notice that our function of interest always lies between y=0..1. Therefore, we will also restrict our y i values to be between 0 and 1. The randomly generated points (x i, y i) have equal probability of landing anywhere in the box which has area 1. The probability of a point landing beneath the function is equal to the area A beneath the curve (which is just the integral of the function of interest) divided by the area of the box. Therefore, if we can determine the probability of a point landing beneath the curve, we can easily approximate the value of the definite integral. The probability will be estimated simply by spraying *n* points into the box and counting how many land beneath the curve *Zn* (i.e. counting the *Hits*). Then the probability *p* is simply $p = Zn / n$.

```
> n := 10000 :
  xH := NULL :
  yH := NULL :
  xM := NULL :
  yM := NULL :
  for i from 1 to n do:
     x := X( ) :
     y := X( ) :
     if y ≤ fcn then;
```

```
        xH := xH, x :
        yH := yH, y :
        else;
        xM := xM, x :
        yM := yM, y :
    end if;
 end do:
xH := [xH] :
yH := [yH] :
xM := [xM] :
yM := [yM] :
 x :='x':
IntEst := evalf( nops(xH) / n );
Hit := ScatterPlot(xH, yH, color = red) :
Miss := ScatterPlot(xM, yM, axes = boxed, view = [0 .. 1 , 0 .. 1], labels = [typeset("x"),
    typeset("f(x)") ], labeldirections = ["horizontal", "vertical"], symbol = circle, symbolsize
     = 20, thickness = 2, tickmarks = [8, 8], colour = blue , axesfont = [Times, 12] , labelfont
     = [Times, 14], axis = [gridlines = [thickness = 2]]) :
display(Hit, Miss);
```

```
                                   IntEst := 0.4770000000
```



If you don't need to keep track of the actual hit and miss points and instead just count the number of hits *Zn* the calculation can be done more compactly and the loop will complete in a shorter time.

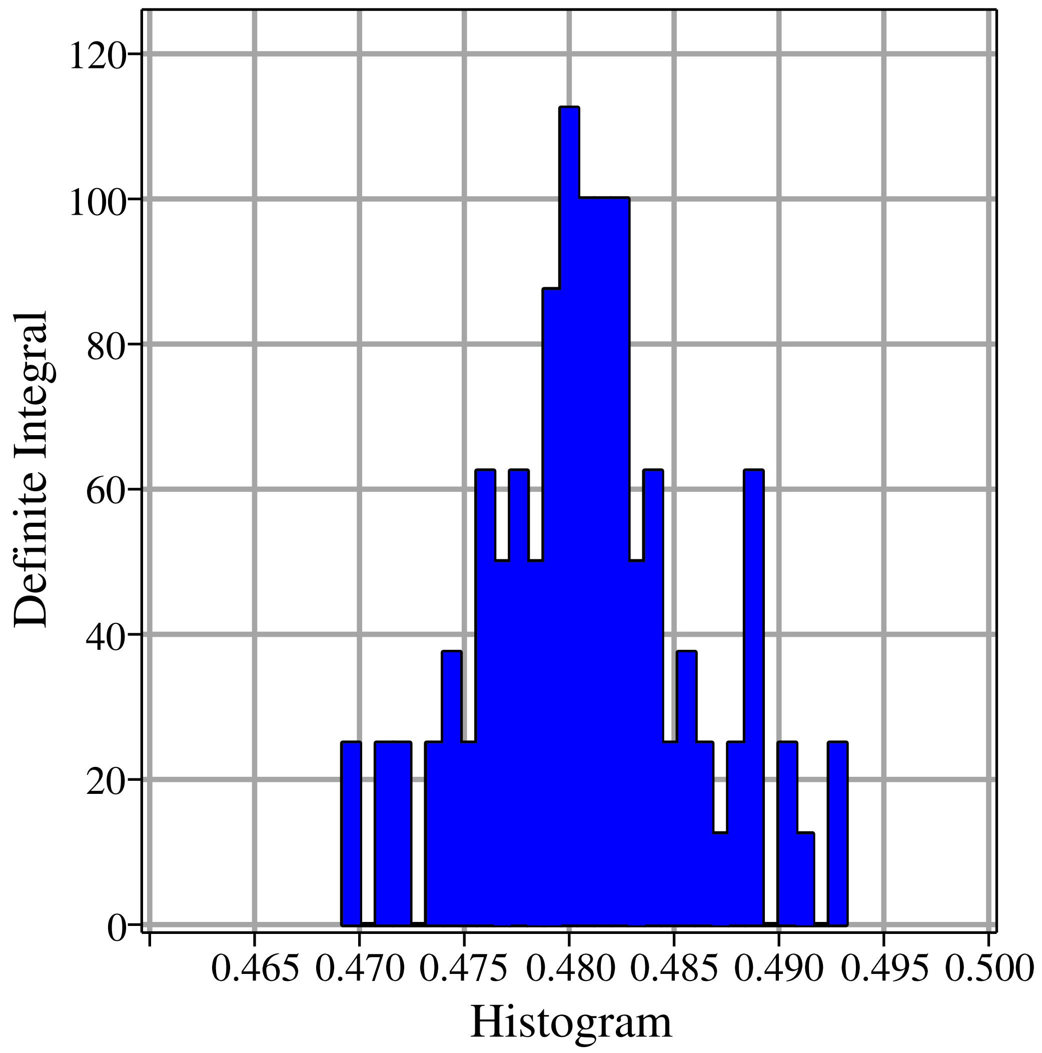
```
> n := 10000 :
  Zn := 0 :
  for i from 1 to n do:
     x := X( ) :
     y := X( ) :
     if y ≤ fcn then;
        Zn := Zn + 1 :
     end if;
  end do:
  x :='x':
  IntEst := evalf(Zn/n);
```

$$IntEst := 0.4840000000 \quad (4)$$

Now we will numerically approximate the integral using n = 1000 one hundred times and plot the resulting distribution of our determination of the integral. The distribution is expected to be Gaussian. WARNING: Depending on the machine that your working with, this chunk of could could take some time to fully execute!

```
> FormatTime("%M:%S");
  n := 10e3 :
  intList := NULL :
  for j from 1 to 100 do:
     Zn := 0 :
     for i from 1 to n do:
        x := X( ) :
        y := X( ) :
        if y ≤ fcn then:
           Zn := Zn + 1 :
        end if:
     end do:
     intList := intList, evalf(Zn/n) :
  end do:
  intList := [intList] :
  FormatTime("%M:%S");
  Histogram(intList, axes = boxed, view = [0.46 .. 0.5 , 0 .. 125], labels
      = [typeset("Histogram"), typeset("Definite Integral") ], labeldirections = ["horizontal",
     "vertical"], symbol = circle, symbolsize = 20, thickness = 2, tickmarks = [8, 8], colour
      = blue , axesfont = [Times, 12] , labelfont = [Times, 14], axis = [gridlines = [thickness
      = 2]]);
  x :='x':
```

```
                              "32:39"
                              "37:31"
```



The width of the distribution can be calculated from the standard deviation of the list of 100 approximations of the integral and is an estimate in the uncertainty of our determination of the definite integral.

```
> StandardDeviation(intList);
                                0.00499948229643062                              (5)
```

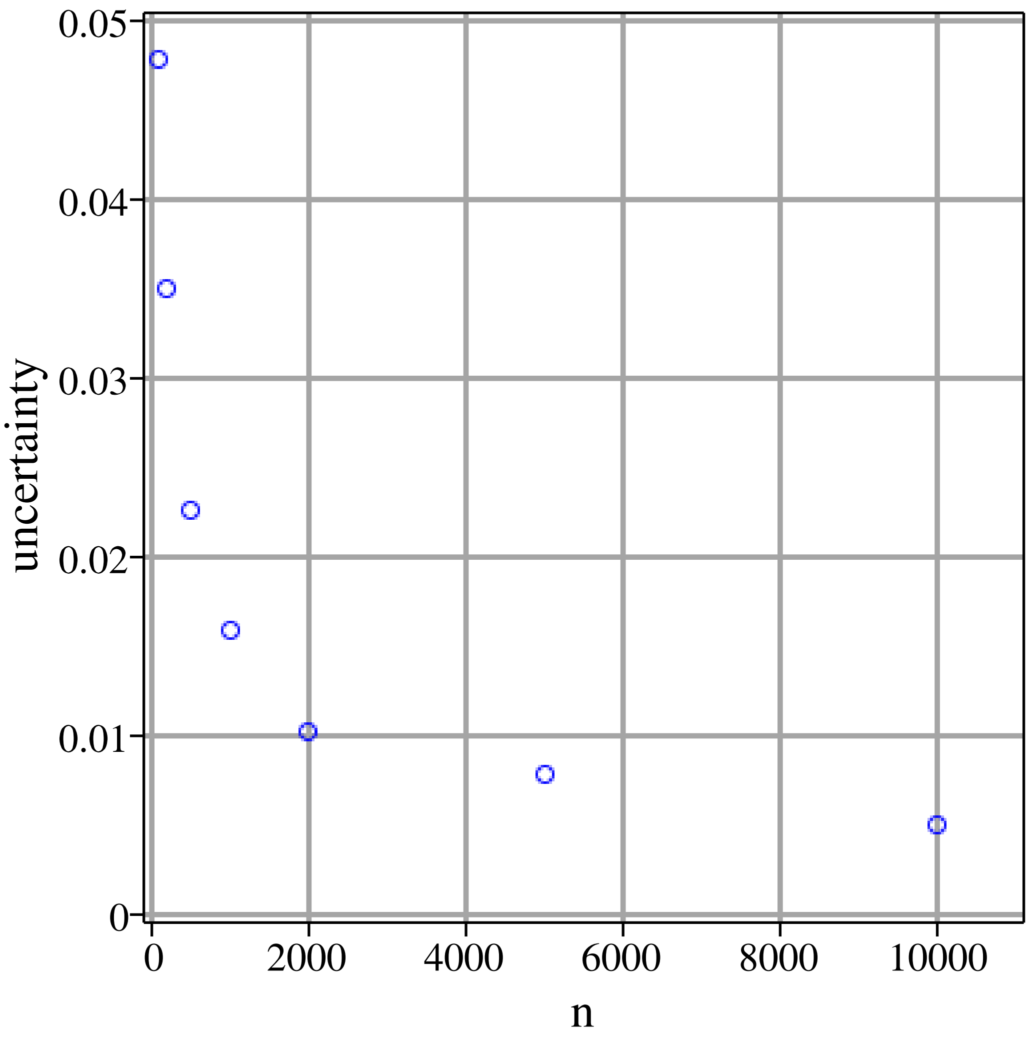
Finally, the last thing we'll attempt to do is to understand how our uncertainly (standard deviation) depends on the *n*. So far, all of our calculations have used *n* = 10e3. Now we'll determine the standard deviation for values of *n* that range from 100 to 10e3. Again, this block of code could take some time to complete. It took my laptop about 9 minutes...

```
> FormatTime("%M:%S");
  nList := [100, 200, 500, 1000, 2000, 5000, 10e3] :
   sigmaList := NULL :
  for k from 1 to nops(nList) do:
      n := nList[k] :
     intList := NULL :
     for j from 1 to 100 do:
        Zn := 0 :
        for i from 1 to n do:
            x := X( ) :
            y := X( ) :
            if y ≤ fcn then:
               Zn := Zn + 1 :
            end if:
         end do:
         intList := intList, evalf(Zn/n) :
      end do:
      intList := [intList] :
      sigmaList := sigmaList, StandardDeviation(intList) :
      print(k);
   end do:
  FormatTime("%M:%S");
   sigmaList := [sigmaList];
   x :='x':
```

```
                              "37:31"
                                 1
                                 2
                                 3
                                 4
                                 5
                                 6
                                 7
                              "46:31"
sigmaList := [0.0479115346395328, 0.0350671793947998, 0.0226086244382514,        (6)
    0.0159154172125805, 0.0102079390771115, 0.00776890330650721,
    0.00498521804838054]
```

Below we plot the uncertainty in the numerical integral estimation as a function of *n* (the number of trials in the Monte Carlo simulation). As expected, the uncertainty decreases as the number of trials increases. The sigma values are proportional to 1/sqrt(*n*).

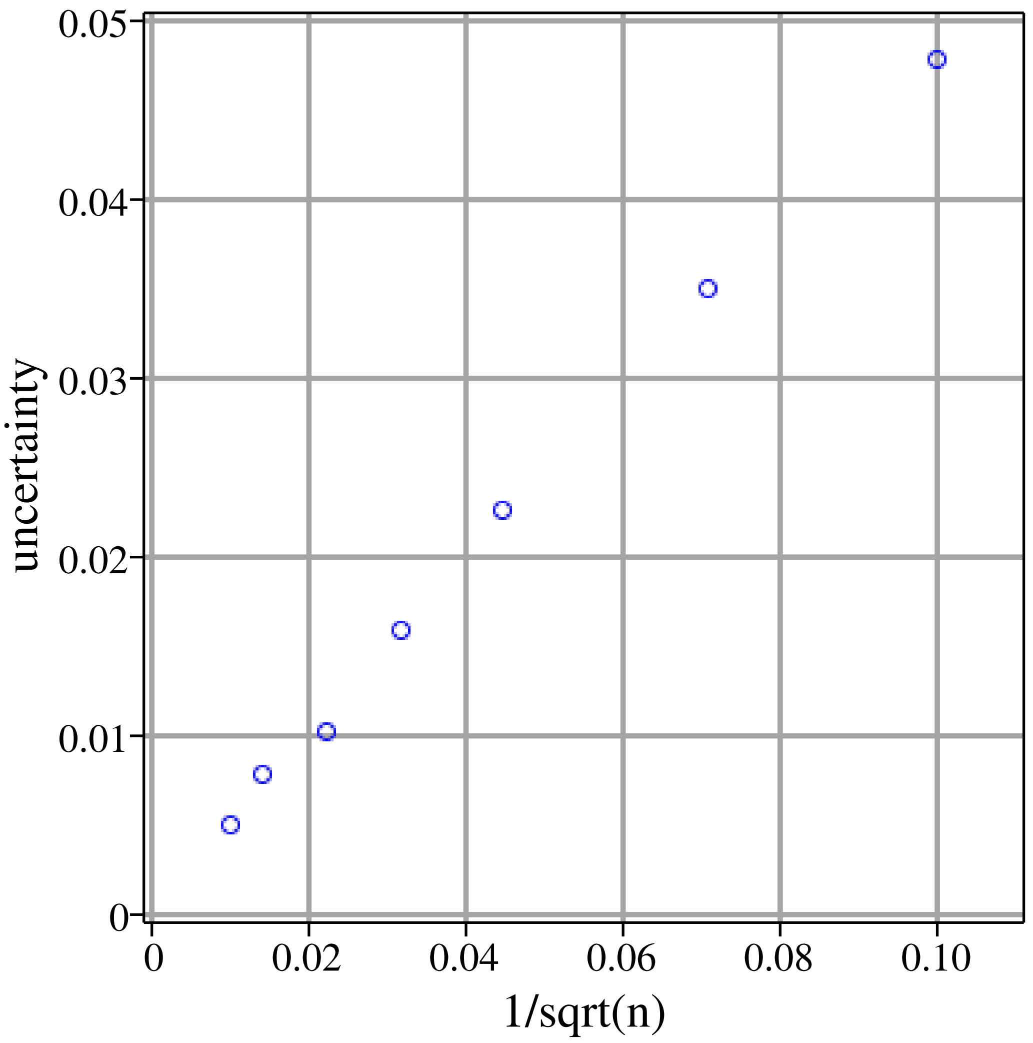
```
> ScatterPlot(nList, sigmaList, axes = boxed, view = [0 .. 11e3 , 0 .. 0.05 ], labels = [typeset("n"),
      typeset("uncertainty") ], labeldirections = ["horizontal", "vertical"], symbol = circle,
      symbolsize = 15, thickness = 2, tickmarks = [8, 8 ], colour = blue , axesfont = [Times, 12 ] ,
      labelfont = [Times, 14 ], axis = [gridlines = [thickness = 2 ]]);
```



To get a better appreciation of the dependence of sigma on *n*, below we plot sigma as a function of 1/sqrt(*n*) and observe the linear relationship between the two. Beautiful! All of this generated from uniformly distributed random numbers! Take a moment to reflect on what we've accomplished. We've used Monte Carlo simulations to study the behaviour of Monte Carlo simulations! The objective of a Monte Carlo calculation is always to study the characteristics of some system (often a physical system) by simulating data using random numbers.

$$> \ SqrtInvn := \left[seq\left(\frac{1}{\mathrm{sqrt}(nList[i])}, i = 1 \, ..nops(nList) \right) \right]:$$

```
ScatterPlot(SqrtInvn, sigmaList, axes = boxed, view = [0 .. 0.11 , 0 .. 0.05 ], labels
    = [typeset("1/sqrt(n)"), typeset("uncertainty") ], labeldirections = ["horizontal",
    "vertical"], symbol = circle, symbolsize = 15, thickness = 2, tickmarks = [8, 8], colour
    = blue , axesfont = [Times, 12 ] , labelfont = [Times, 14 ], axis = [gridlines = [thickness
    = 2 ]]);
```


uncertainty
1/sqrt(n)
0
0.01
0.02
0.03
0.04
0.05
0.02
0.04
0.06
0.08
0.10

>