CODING TAKES NUMERICAL ANALYSIS AND SIMULATION TO THE NEXT LEVEL

L.A.A. Blomme – European School Brussels 3

TI-symposium, Brussels – 7/10/2017

Programming can be done in many ways: in a notes page, a spreadsheet or by coding a programme or a function in TI-basic in the programme editor of the TInspire CAS CX.

1. THE SUM OF TWO DICE THROWS

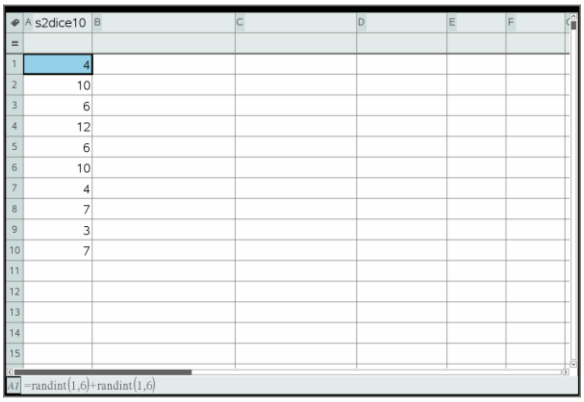
TNS-file: 1.1 sum of 2 dice

We are going to simulate the throwing of two dice and observe the distribution of the sums obtained. We are going to throw the two dice 10 times, 20 times, 50 times and 100 times, first in a spreadsheet and then using a program.

1.1. <u>First method: in a spreadsheet</u>

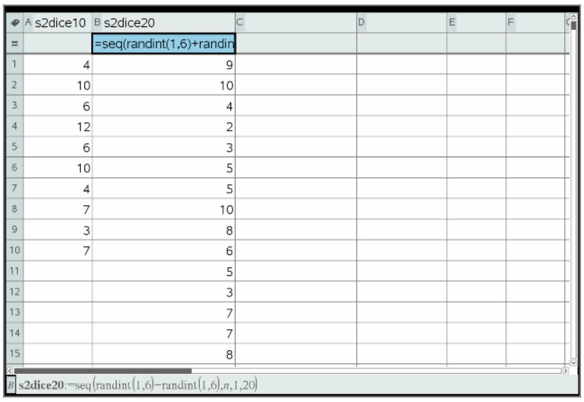
We create 4 lists in the spreadsheet.

- In the cell A1, write "=randint(1,6)+randint(1,6)enter".
  Then, select the cell A1 then "ctrl menu Fill" (10 cells).
  Name the column "s2dice10".
- Place the cursor in the second column second row (grey) and write "seq(randint(1,6)+randint(1,6),n,1,20)".
  Name the column "s2dice20".
  Repeat the operation for n = 50 and n = 100.



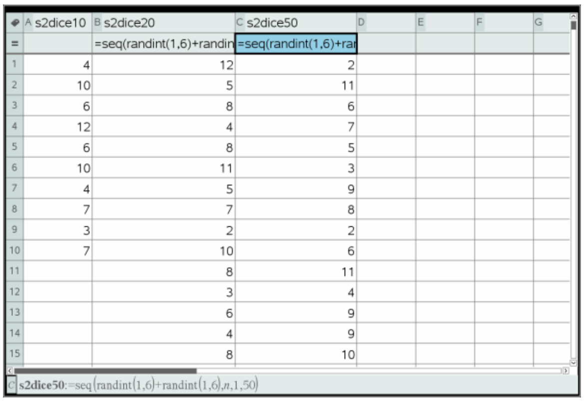
	A s2dice10	B	C	D	E	F
=						
1	4					
2	10					
3	6					
4	12					
5	6					
6	10					
7	4					
8	7					
9	3					
10	7					
11						
12						
13						
14						
15						

A1 =randint(1,6)+randint(1,6)



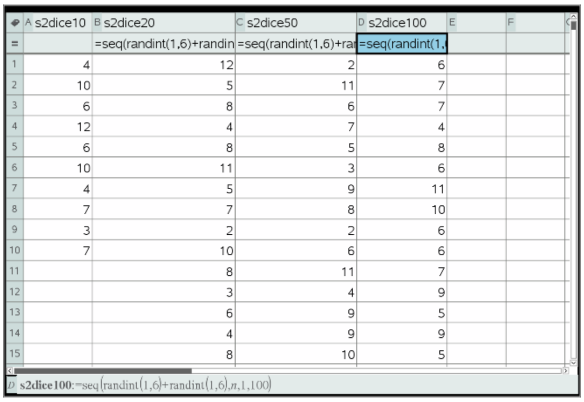
	A s2dice10	B s2dice20	C	D	E	F
=		=seq(randint(1,6)+randin				
1	4	9				
2	10	10				
3	6	4				
4	12	2				
5	6	3				
6	10	5				
7	4	5				
8	7	10				
9	3	8				
10	7	6				
11		5				
12		3				
13		7				
14		7				
15		8				

B s2dice20:=seq(randint(1,6)+randint(1,6),n,1,20)



	A s2dice10	B s2dice20	C s2dice50	D	E	F	G
=		=seq(randint(1,6)+randin	=seq(randint(1,6)+ra				
1	4	12	2				
2	10	5	11				
3	6	8	6				
4	12	4	7				
5	6	8	5				
6	10	11	3				
7	4	5	9				
8	7	7	8				
9	3	2	2				
10	7	10	6				
11		8	11				
12		3	4				
13		6	9				
14		4	9				
15		8	10				

C s2dice50:=seq(randint(1,6)+randint(1,6),n,1,50)

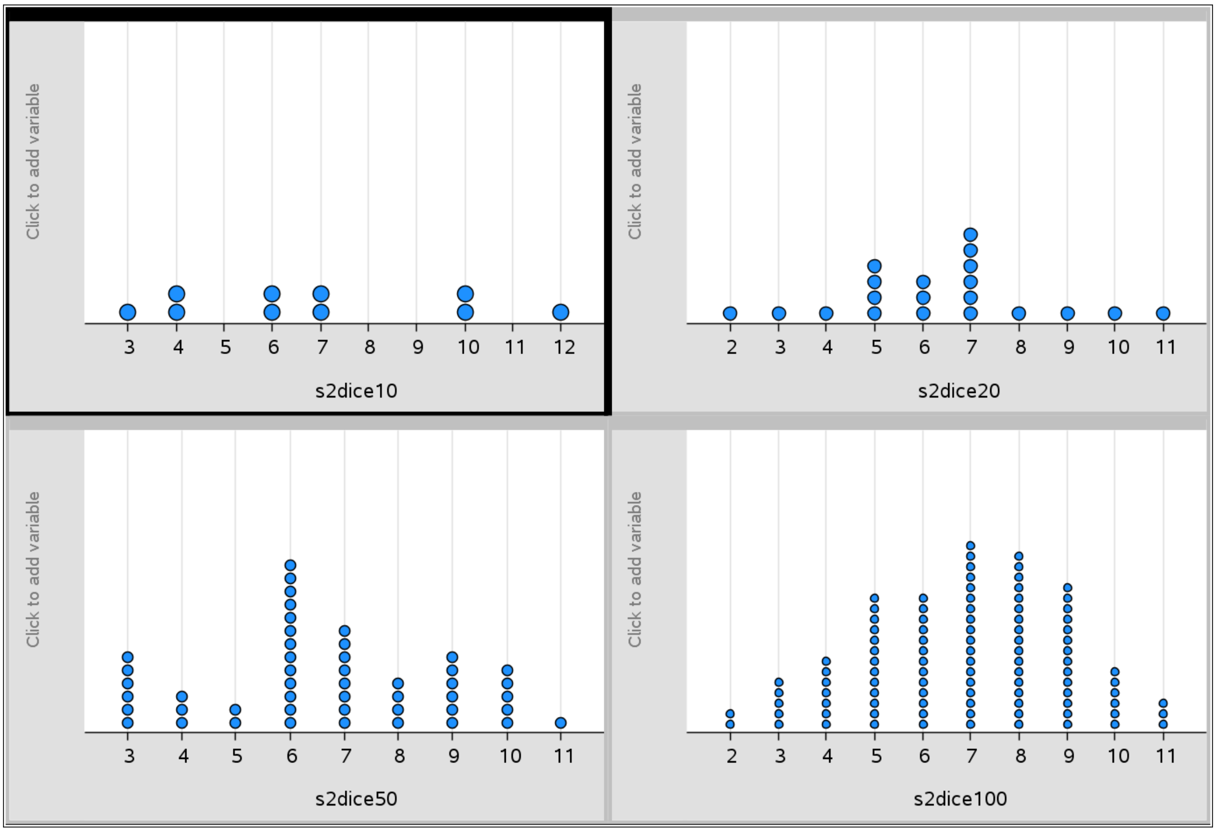


	A s2dice10	B s2dice20	C s2dice50	D s2dice100	E	F
=		=seq(randint(1,6)+randin	=seq(randint(1,6)+ra	=seq(randint(1,		
1	4	12	2	6		
2	10	5	11	7		
3	6	8	6	7		
4	12	4	7	4		
5	6	8	5	8		
6	10	11	3	6		
7	4	5	9	11		
8	7	7	8	10		
9	3	2	2	6		
10	7	10	6	6		
11		8	11	7		
12		3	4	9		
13		6	9	5		
14		4	9	9		
15		8	10	5		

D s2dice100:=seq(randint(1,6)+randint(1,6),n,1,100)

We add a data and statistics page and split it into 4 parts and make a quick graph of the data in the spreadsheet.
We can graph the four results on the same page.

- “ctrl +page doc 5: Page Layout – 2: Select Layout – 8: Layout 8”.
- In menu, choose “5: Add Data & Statistics”.
- Move the cursor to the middle bottom of the page and choose s2dice10.
- Do the same for the other three graphs.

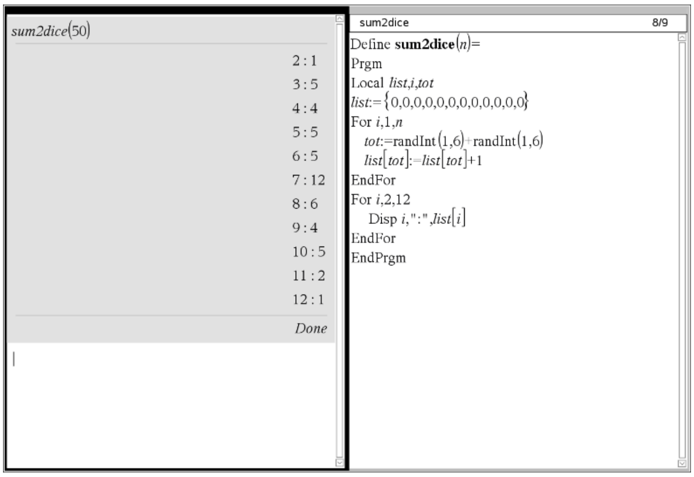


1.2. <u>Second method: programming</u>

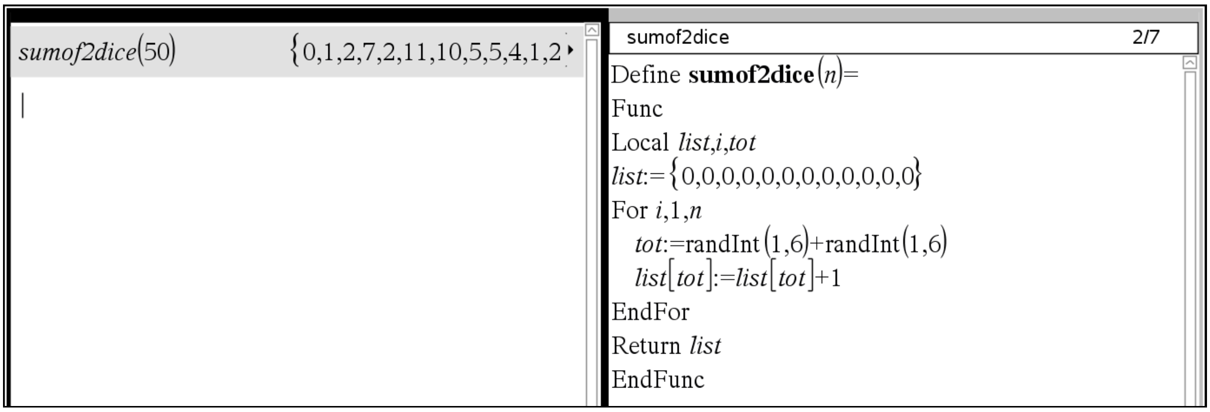
The first possibility would be to write a program that displays the results. There is however nothing much we can do with the displayed data.

- Go to the page containing the program **sum2dice()**.
- Type in the left window: sum2dice(50)

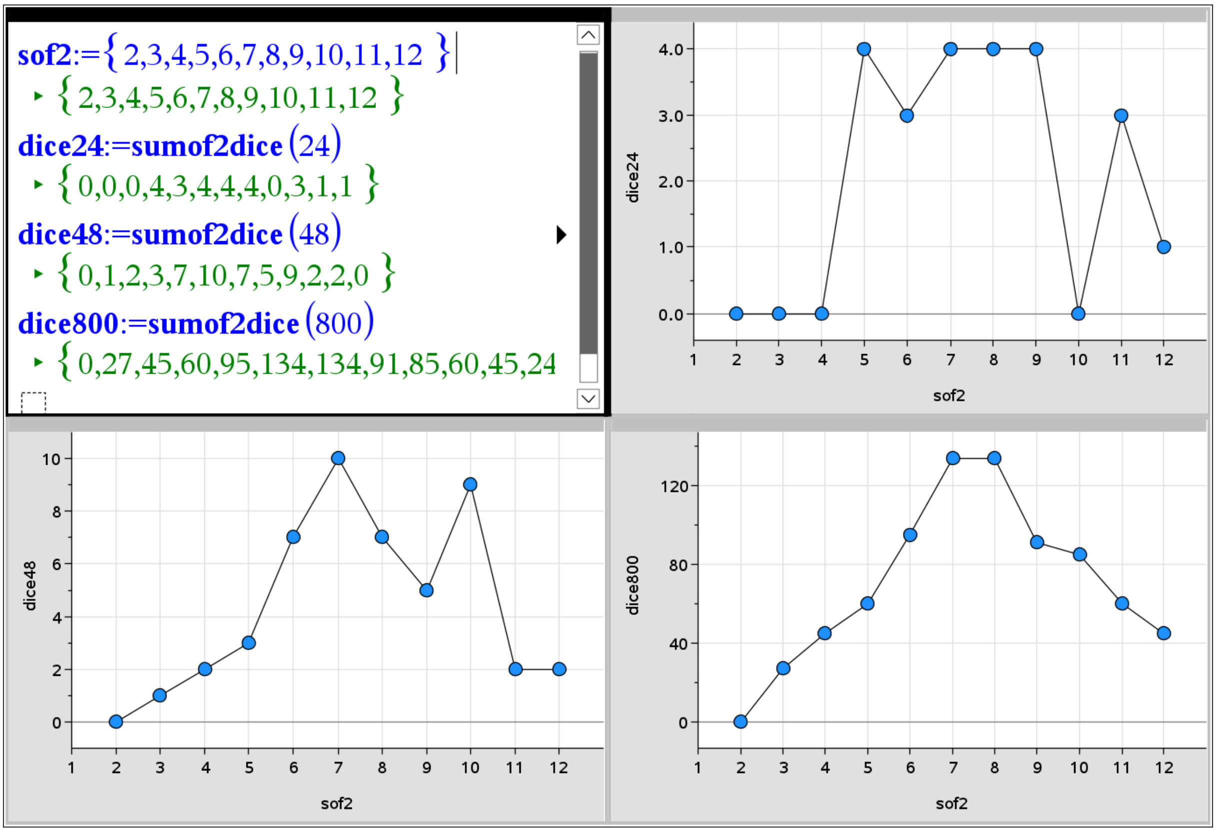
The program is executed and the results are displayed in the calculator window.



A more clever solution is to write a function in TI-basic that generates a list containing the results of the simulation.



In a calculator window, we define the list of possible sums. Applying the new function **sumof2dice()** 3 more lists of simulation data for n=24, 48 and 800 can be created.



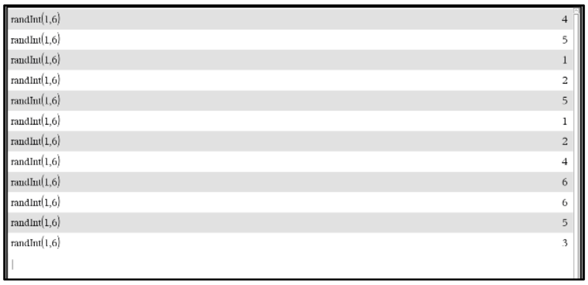
2. TOYS IN BOXES OF CEREAL

TNS-file: 2.1 cereal

In this problem, every box of cereals bought from a shop contains one toy from a selection of six different toys; the toys being evenly distributed among the cereal boxes. The simulation allows an investigation of the number of boxes of cereal one might have to buy in order to complete the set of 6 toys.

The simulation uses the function randint(1,6) to generate random integers 1-6. In the part of the investigation the randint function is used to simulate manually the number of boxes which must bought and in the second part a simple program is used to automate the simulation. The simulation is run 60 times in order to obtain some summary statistics for the number of cereal boxes which must be bought. Later the effect of the introduction of a bias in the distribution is considered.

2.1 First method: a manual simulation

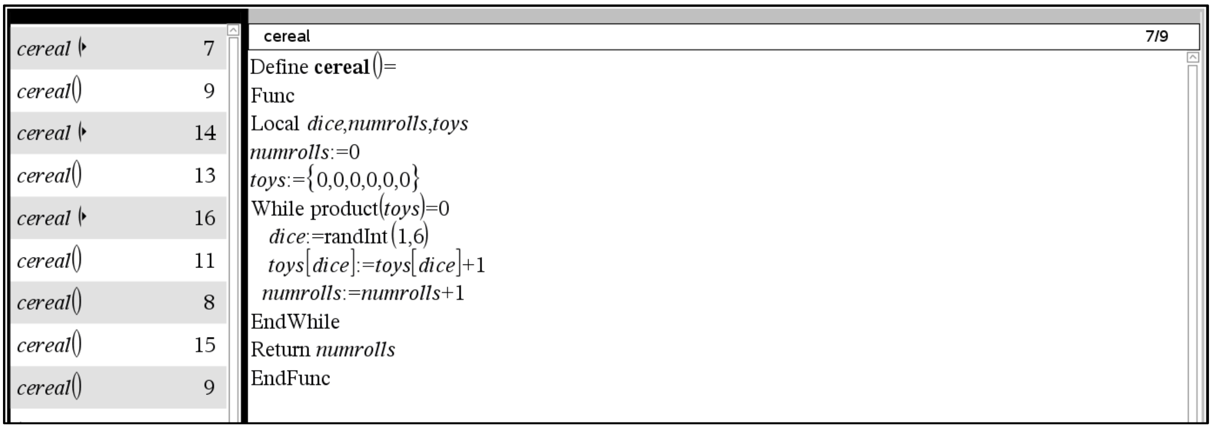


Use RANDINT(1,6) to simulate the choice of a toy. Type the command only once and press enter to repeat the same instruction. How many boxes of cereals do you have to buy to collect all 6 toys? What is the average number of boxes a customer has to buy?

Toy 1	*Toy 2*	*Toy 3*	*Toy 4*	*Toy 5*	*Toy 6*	*sum*

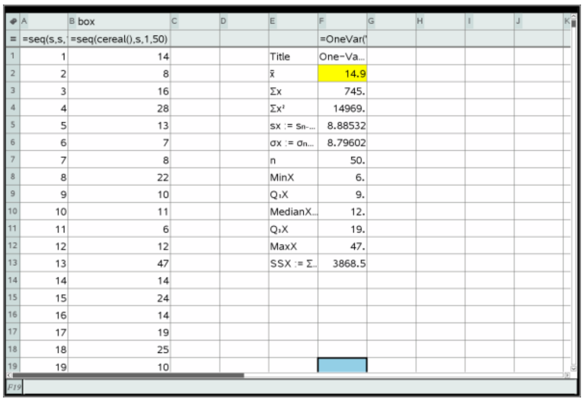
2.2 Second method: programming

We code a function **cereal()** to simulate this problem. Once the new function is defined we can use it in any application: a calculator sheet as well as a spreadsheet.

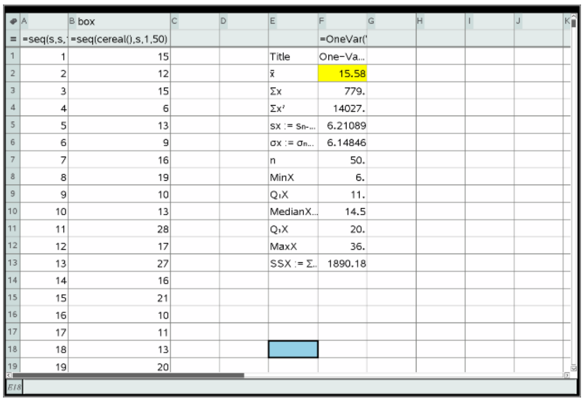


Using the command =SEQ(**cereal()**,s,1,50) we create the list “box” containing 50 simulations.

- Click Statistics; Stat Calculations and 1. One variable statistics.
  To repeat all calculations in the spreadsheet, press ctrl+R.



	A	B box	E	F
=	=seq(s,s,	=seq(cereal(),s,1,50)		=OneVar('
1	1	14	Title	One-Va...
2	2	8	x̄	14.9
3	3	16	Σx	745.
4	4	28	Σx²	14969.
5	5	13	sx := sn-...	8.88532
6	6	7	σx := σn...	8.79602
7	7	8	n	50.
8	8	22	MinX	6.
9	9	10	Q₁X	9.
10	10	11	MedianX...	12.
11	11	6	Q₃X	19.
12	12	12	MaxX	47.
13	13	47	SSX := Σ..	3868.5
14	14	14		
15	15	24		
16	16	14		
17	17	19		
18	18	25		
19	19	10		



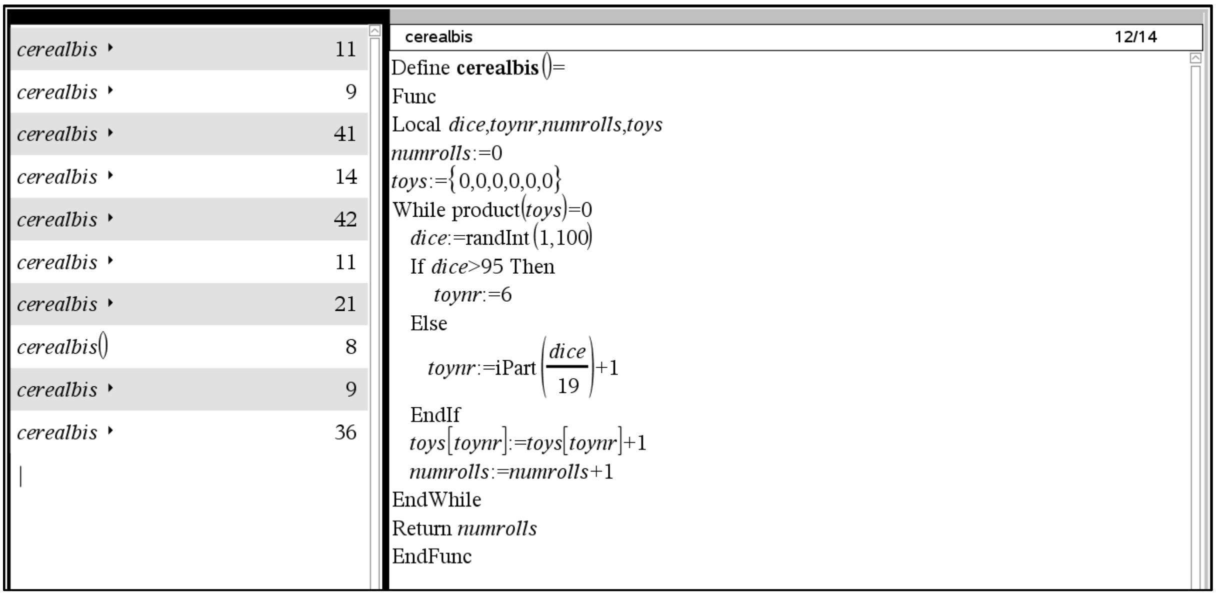
	A	B box	E	F
=	=seq(s,s,	=seq(cereal(),s,1,50)		=OneVar('
1	1	15	Title	One-Va...
2	2	12	x̄	15.58
3	3	15	Σx	779.
4	4	6	Σx²	14027.
5	5	13	sx := sn-...	6.21089
6	6	9	σx := σn...	6.14846
7	7	16	n	50.
8	8	19	MinX	6.
9	9	10	Q₁X	11.
10	10	13	MedianX...	14.5
11	11	28	Q₃X	20.
12	12	17	MaxX	36.
13	13	27	SSX := Σ..	1890.18
14	14	16		
15	15	21		
16	16	10		
17	17	11		
18	18	13		
19	19	20		

<u>2.3 a new distribution of the toys</u>

The 6 different toys are no longer uniformly distributed. Toy_6 appears in only 5 % of the boxes. The other toys appear in 19% of the boxes.

How many boxes of cereal do you have to buy to have all 6 toys?

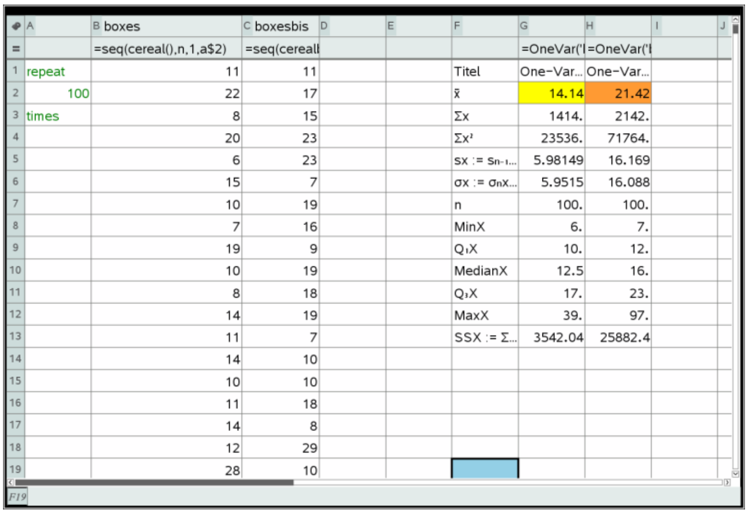
We code a function **cerealbis()** to simulate the new problem.



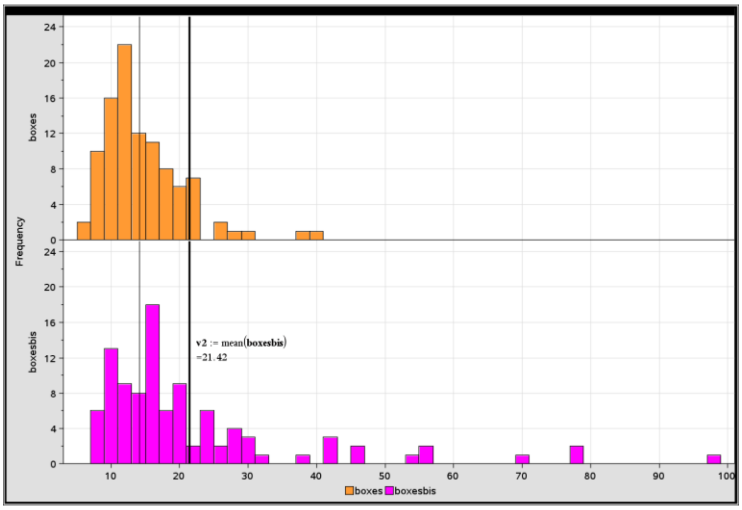
cerealbis ▸	11
cerealbis ▸	9
cerealbis ▸	41
cerealbis ▸	14
cerealbis ▸	42
cerealbis ▸	11
cerealbis ▸	21
cerealbis()	8
cerealbis ▸	9
cerealbis ▸	36

```
cerealbis                                    12/14
Define cerealbis()=
Func
Local dice,toynr,numrolls,toys
numrolls:=0
toys:={0,0,0,0,0,0}
While product(toys)=0
  dice:=randInt(1,100)
  If dice>95 Then
    toynr:=6
  Else
    toynr:=iPart(dice/19)+1
  EndIf
  toys[toynr]:=toys[toynr]+1
  numrolls:=numrolls+1
EndWhile
Return numrolls
EndFunc
```

We can now compare the two types of distribution of the toys. What happens to the average number of boxes of cereals a customer has to buy to get all the toys, if they are no longer uniformly distributed?



	A	B boxes	C boxesbis	D	E	F	G	H
=		=seq(cereal(),n,1,a$2)	=seq(cereal				=OneVar('	=OneVar('
1	repeat	11	11			Titel	One-Var...	One-Var...
2	100	22	17			x̄	14.14	21.42
3	times	8	15			Σx	1414.	2142.
4		20	23			Σx²	23536.	71764.
5		6	23			sx := sn-1...	5.98149	16.169
6		15	7			σx := σnx...	5.9515	16.088
7		10	19			n	100.	100.
8		7	16			MinX	6.	7.
9		19	9			Q₁X	10.	12.
10		10	19			MedianX	12.5	16.
11		8	18			Q₃X	17.	23.
12		14	19			MaxX	39.	97.
13		11	7			SSX := Σ...	3542.04	25882.4
14		14	10					
15		10	10					
16		11	18					
17		14	8					
18		12	29					
19		28	10					

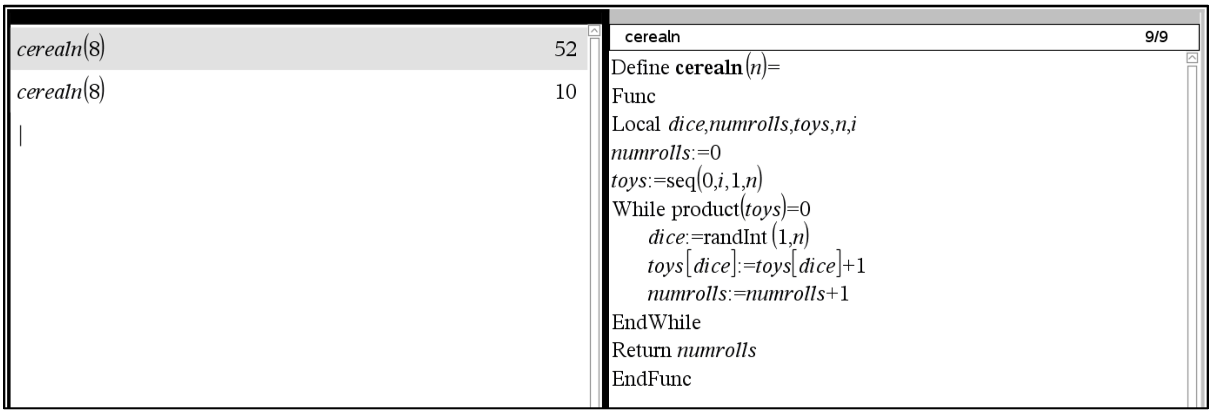


2.4 a variable number of toys

TNS-file: 2.2 cereal 2

In this extended problem, every box of cereals bought from a shop contains one toy from a selection of a variable number n of different toys; the toys being evenly distributed among the cereal boxes.

The function cereal(n) is a more flexible and general version of the function cereal().



```
cerealn(8)        52
cerealn(8)        10
```

```
cerealn                                  9/9
Define cerealn(n)=
Func
Local dice,numrolls,toys,n,i
numrolls:=0
toys:=seq(0,i,1,n)
While product(toys)=0
    dice:=randInt(1,n)
    toys[dice]:=toys[dice]+1
    numrolls:=numrolls+1
EndWhile
Return numrolls
EndFunc
```

We can still extend the complexity of the problem with a variable number of toys n, a variable probability of toy_n and a variable number k of simulations.

define the global variables

number of toys

n:=9 ▸ 9

probability of toy_n

p:=4 ▸ 4

number of simulations

k:=75 ▸ 75

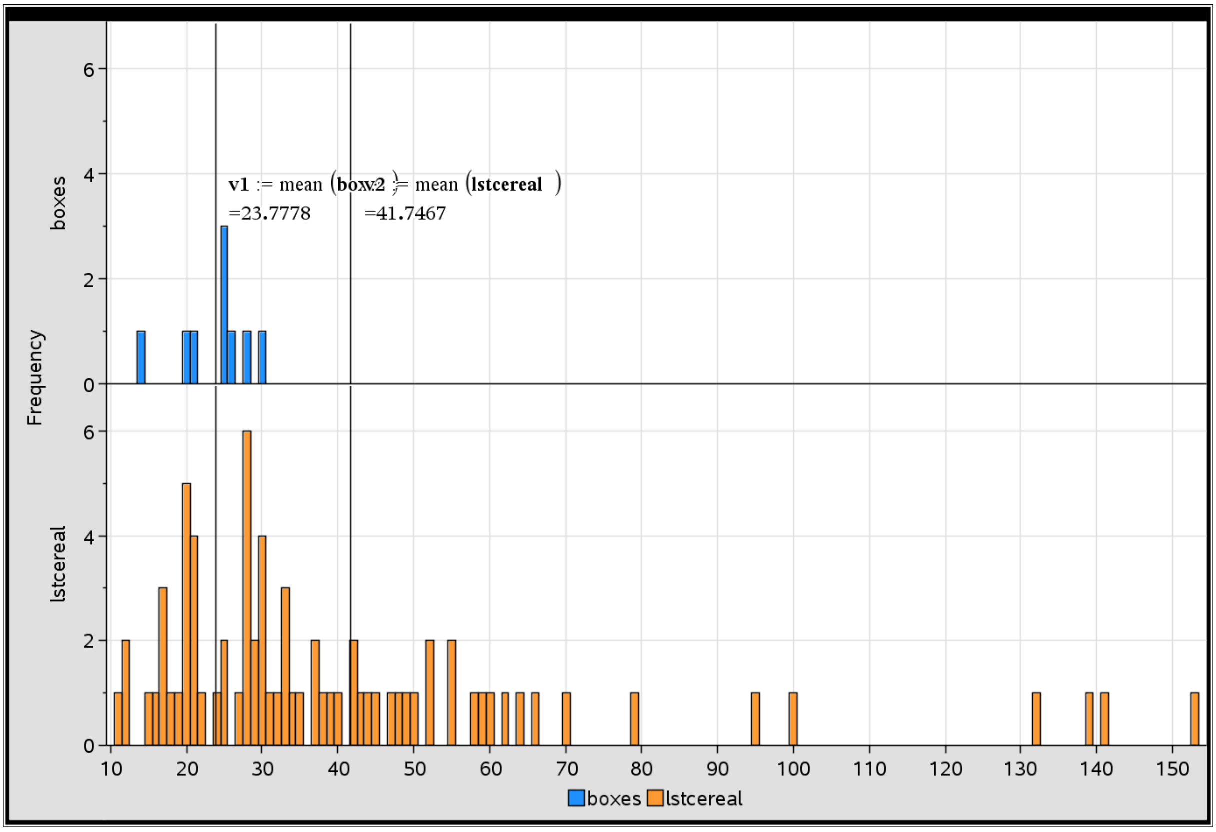
output list

cereal3(n,p,k) ▸ *Done*

```
* cereal3                                        17/17
Define cereal3(n,p,k)=
Prgm
Local dice,toys,n,i,p,k,x,numrolls
lstcereal:=seq(0,i,1,k)
For x,1,k
  numrolls:=0
  toys:=seq(0,i,1,n)
  While product(toys)=0
    dice:=randInt(0,9900)
    If dice≥(100−p)· 100 Then
      toys[n]:=toys[n]+1
    Else
      dice:=randInt(1,n−1)
      toys[dice]:=toys[dice]+1
    EndIf
    numrolls:=numrolls+1
  EndWhile
  lstcereal[x]:=numrolls
gloEndFor
EndPrgm
```

	A	B	C boxes	D lstcereal	E	F
=		=seq(i,i,1,'n)	=seq(cerealn('n),i,1,'n)			
1	number_of_toys	1	14	33		
2	9	2	21	29		
3	probability_of	3	24	37		
4	toy_n_(in%)	4	30	48		
5	4	5	16	42		
6	probability_of	6	22	100		
7	other_toys_(in%)	7	18	25		
8	10.6667	8	17	28		
9		9	44	17		
10				12		
11				139		
12				49		
13				30		
14				44		
15				21		

A1 *number_of_toys*



3. RIEMANN SUM

TNS-file: 3.1 Riemann sum

A Riemann sum is a certain kind of approximation of a definite integral by a finite sum. The sum is calculated by dividing the region up into rectangles and adding these rectangles together.

Because the region filled by the small shapes is usually not the same shape as the region being measured, the Riemann sum will differ from the area being measured. This error can be reduced by dividing up the region more finely, using smaller and smaller shapes. As the shapes get smaller and smaller, the sum approaches the Riemann integral.

We use and compare three different methods of Riemann summation with partitions of equal size. The interval [a,b] is therefore divided into n subintervals.

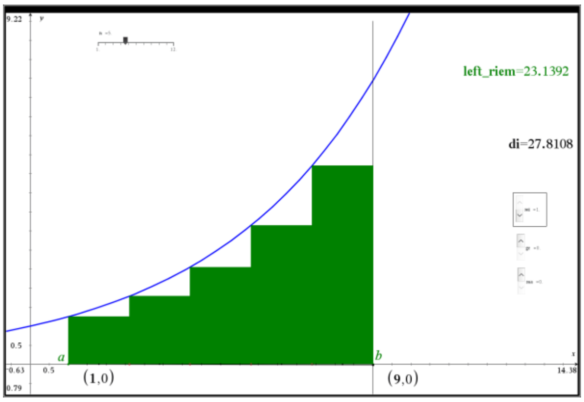
- Left Riemann sum: the function is approximated by its value at the left-end point of each subinterval.
- Right Riemann sum: the function is approximated by its value at the right-end point of each subinterval.
- Middle Riemann sum: the function is approximated by its value at the midpoint of each subinterval.

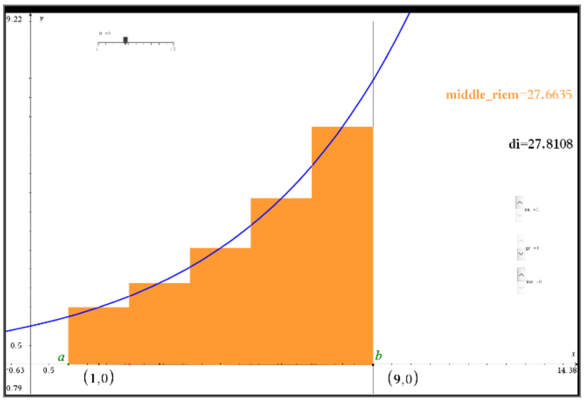
3.1 a special case: a positive monotonically increasing function

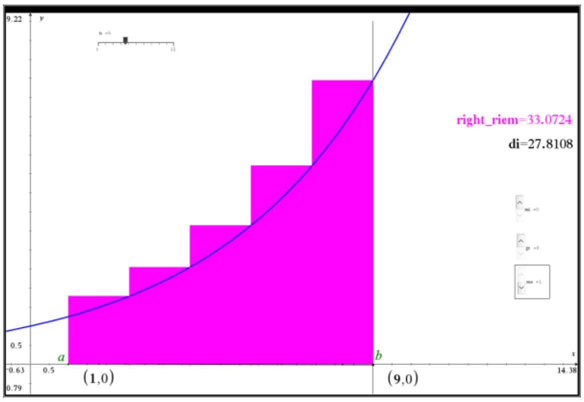
As an introduction to the Riemann sum we use the positive, monotonically increasing function $f(x) = 1.25^x$ in the interval [1,9].

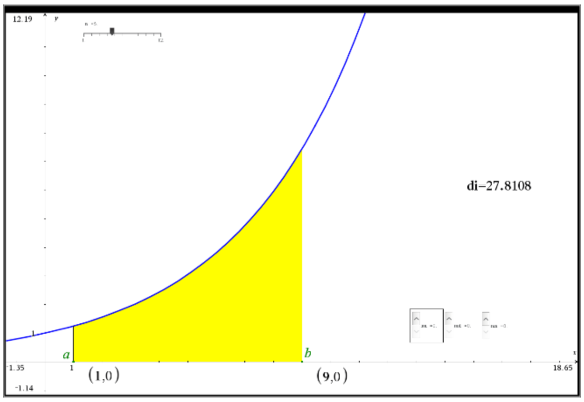
In this case, the left Riemann sum amounts to an underestimation. The rectangles have a height equal to the minimum value of the function in each subinterval. The right Riemann sum amounts to an overestimation due to rectangles with a height of the maximum value of the function in a subinterval.

The third method, using an average value of the function in each subinterval amounts to a better estimation.

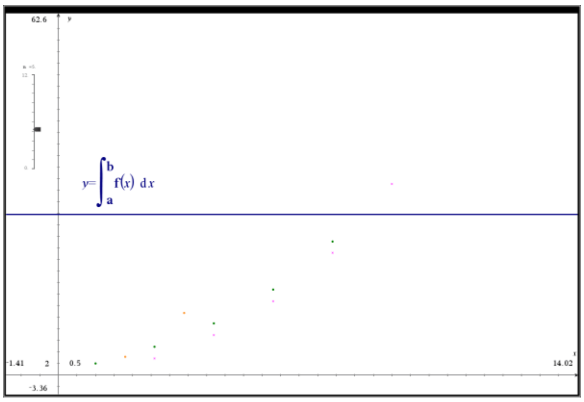


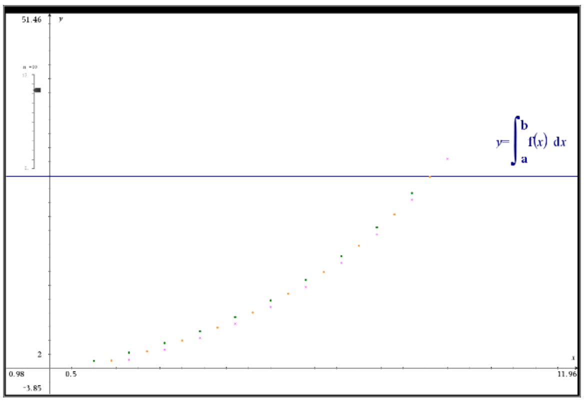






As n increases, the shapes get smaller and smaller and the sum approaches the Riemann integral. To visualise this, we create 3 lists: left Riemann sum, right Riemann sum and middle Riemann sum. Every list is displayed as a dynamic scatterplot, adapting to the value of n.

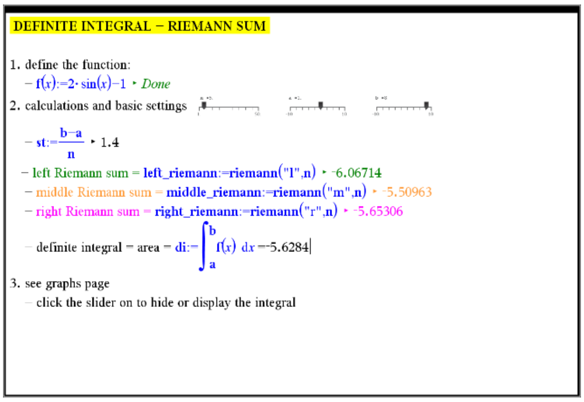


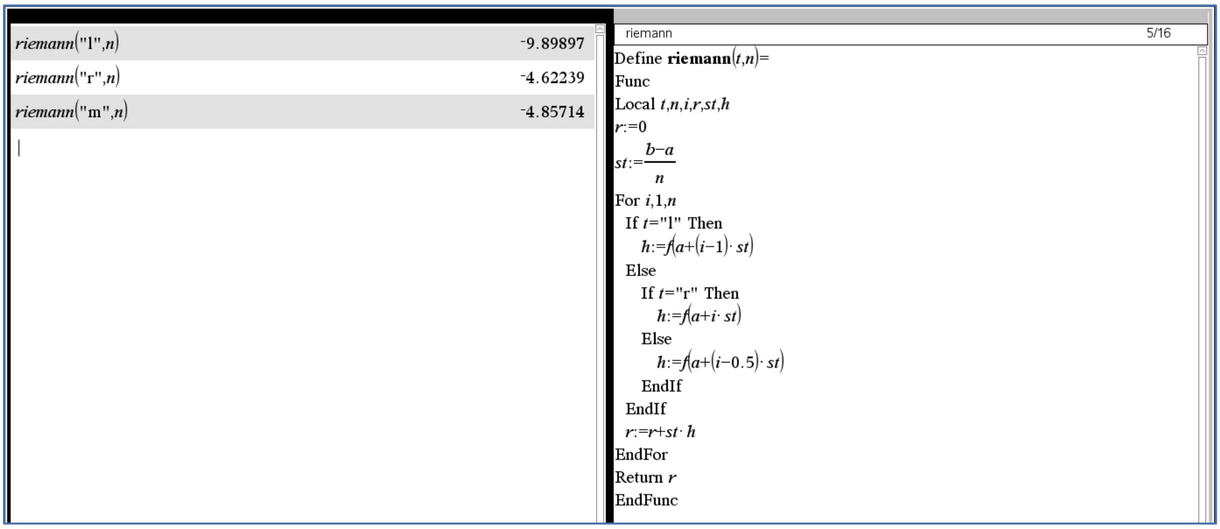


3.2 a more general approach

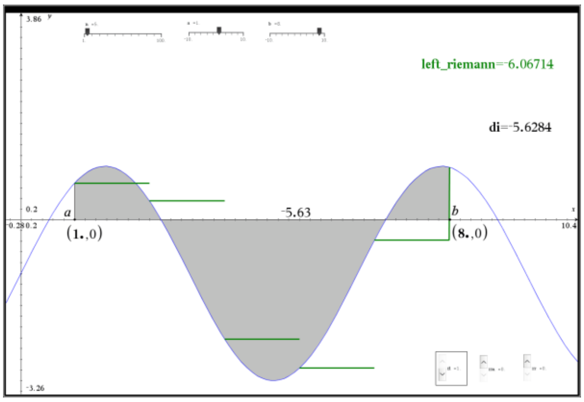
To calculate the three different types of Riemann sum we write one function **Riemann()** with two parameters: l (= left), r (= right) or m (= middle) and n (= number of subintervals).

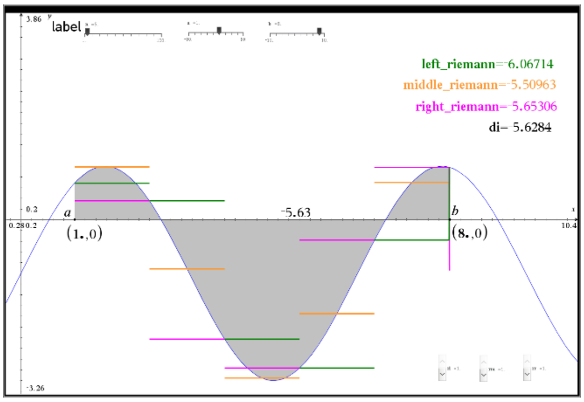
We assume that the function f(x) and the interval [a,b] are predefined: e.g. $f(x) = 2\sin(x) - 1$ in the interval [1,8].



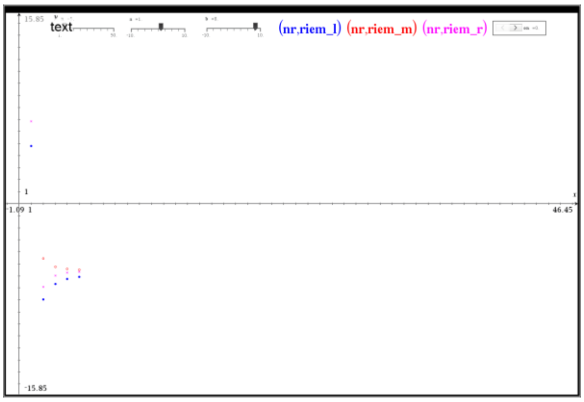


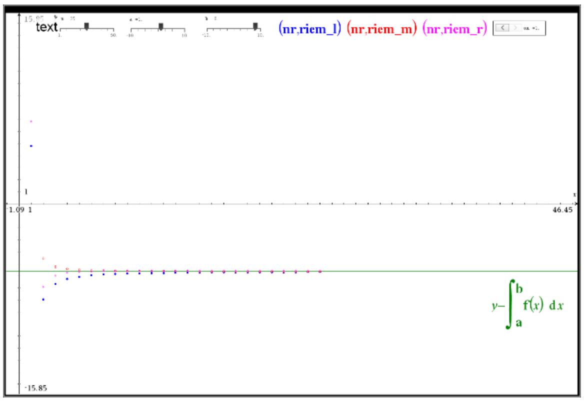
The difference between the 3 types of Riemann approximations can be shown on the next graphs page. Click the sliders rl (Riemann left), rm (Riemann middle) and/or rr (Riemann right) to hide or display the approximating Riemann functions or rectangles.



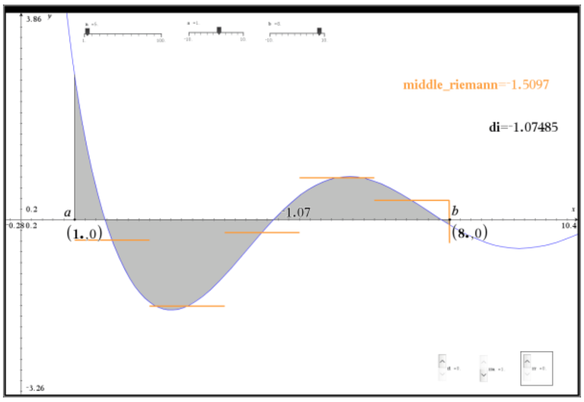


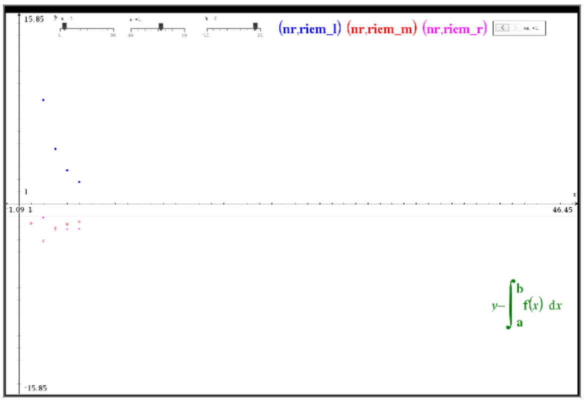
A spreadsheet can be used to create 3 lists, one for each of the different types of Riemann sum, every time using the new function **Riemann()**. These lists can be graphically represented as dynamic scatterplots, automatically adapting to changing values of n, a or b.





To apply this approach to another function all we need to do is change the definition of the function on the first notes page.





4. SOLVING NON-LINEAR EQUATIONS – NEWTON'S METHOD

TNS-file: 4.1 newton

In numerical analysis, **Newton's method** (also known as the **Newton–Raphson method**), is a method for finding successively better approximations to the roots (or zeroes) of a real-valued function. It's a nice example of a root-finding algorithm: $f(x)=0$.

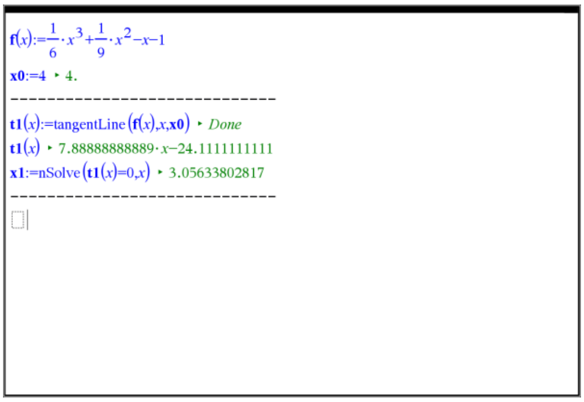
The method starts with a function *f* defined over the real numbers x, the function's derivative *f ′*, and an initial guess x_0 for a root of the function *f*. If the function satisfies the assumptions made in the derivation of the formula and the initial guess is close, then a better approximation x_1 is $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$.

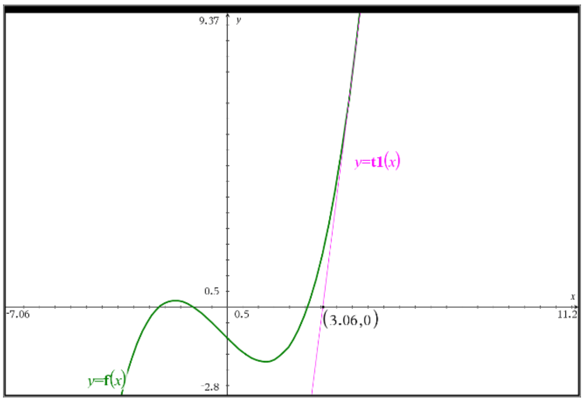
Geometrically, $(x_1, 0)$ is the point of intersection of the *X*-axis and the tangent line of the graph of *f* at $(x_0, f(x_0))$.

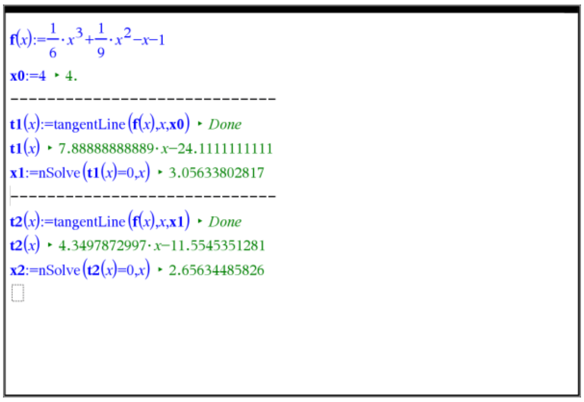
The process is repeated as $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ a fixed number of times or until a sufficiently accurate value is reached.

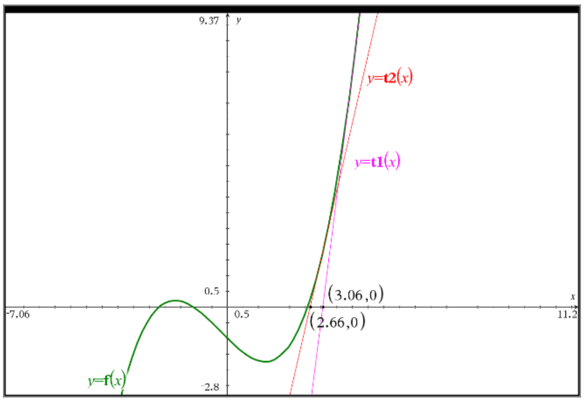
4.1 Newton's method - a manual approach

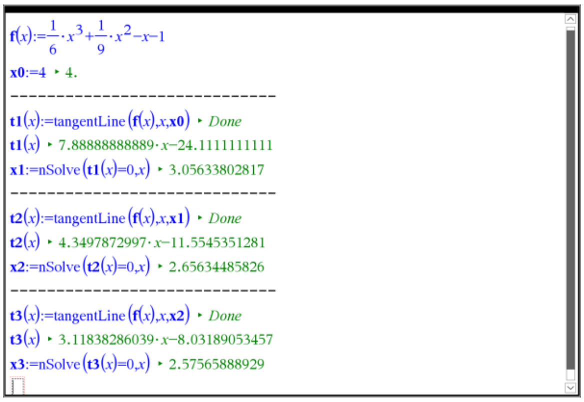
In a notes page we define the function and the value of x_0. We calculate the point of intersection of the tangent line of *f* at $(x_0, f(x_0))$. The tangent line is also displayed in the graphs page. This process is repeated.

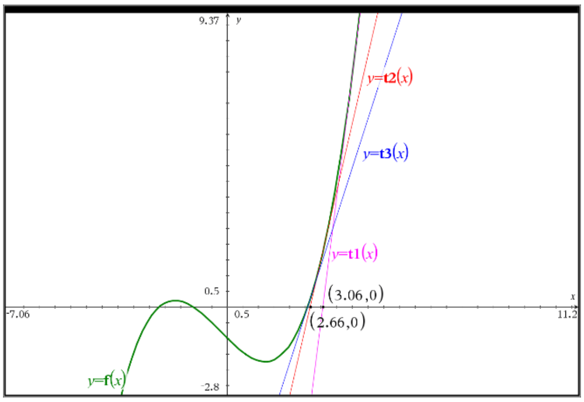






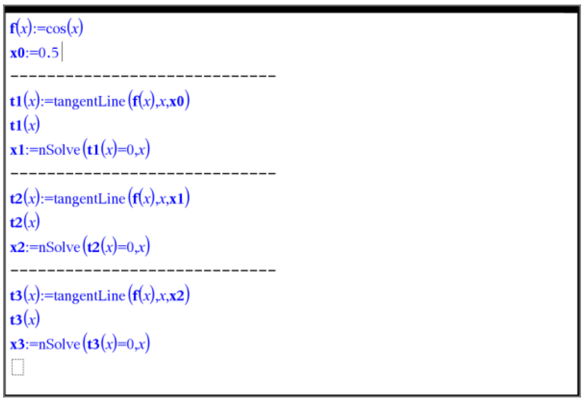


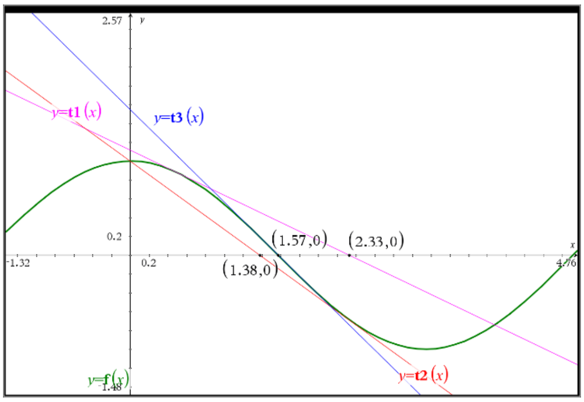




Newton's method can be easily repeated for another equation / function. Using a notes page to perform all the calculations has a big advantage: all calculations in all linked math boxes are automatically updated every time a math box is changed. This is the simplest attempt to programming.

All we need to do to apply Newton's method to another equation, is change the definition of the function in the notes page and chose a proper value x_0 for the approximation process.





4.2 Newton's method – in a spreadsheet

We can use a spreadsheet to calculate a list of approximating values with the general formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- Define as usual, f(x) in a notes page.
- In the first column of the spreadsheet, we count the number of steps performed.
- Cell B3 contains the starting value x_0.
- Cell C3 contains f(a). In step 7 this value is almost 0. Therefore x_7 is an excellent approximation of the root.
- Cell D3 contains the formula for the calculation of $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$ or

$$B3 - \frac{f(B3)}{\frac{d}{dx}(f(x))|x = B3}$$

- On the next line of the spreadsheet the value of D3 is copied into cell A4.
- Repeat the process for columns C and D, etc.

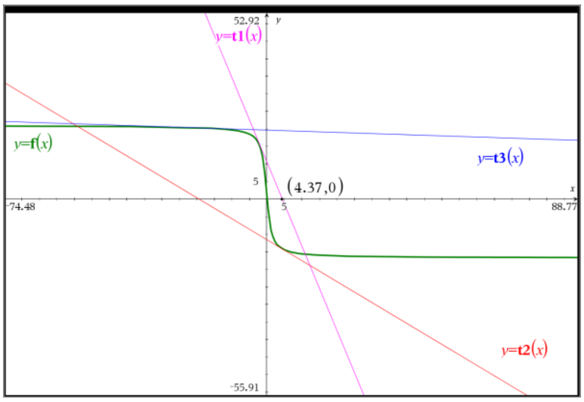
	A st	B a	C fa	D	E	F	G	H	I
=									
1	step_i	x0		x_i+1	x^3+x^2−x−1.				
2									
3	0.	4.	75.	2.63636363636					
4	1.	2.63636363636	21.637866266	1.77511961722					
5	2.	1.77511961722	5.96942009648	1.27780835839					
6	3.	1.27780835839	1.44138391963	1.05447646007					
7	4.	1.05447646007	0.22993824806	1.00274350042					
8	5.	1.00274350042	0.011004129492	1.00000749595					
9	6.	1.00000749595	0.000029984012	1.00000000006					
10	7.	1.00000000006	0.000000000225	1.					
11	8.	1.	0.	1.					
12	9.	1.	0.	1.					
13	10.	1.	0.	1.					

Since f(x) is defined in a notes page, to repeat Newton's method for another function, simply change the definition of the function in the notes page and adapt the starting value of x_0 in the spreadsheet e.g. $x_0 = -1$ and $f(x) = 2 - Arc\tan(x)$.

	A st	B a	C fa	D	E
=					
1	step_i	x0	_	x_i+1	2.−12.*tan⁻¹(x)
2	_	_	_	_	
3	0.	-1.	11.4247779608	0.9041296601…	
4	1.	0.904129660128	-6.82110403104	-0.128955218…	
5	2.	-0.1289552180…	3.53896938327	0.1708631562…	
6	3.	0.170863156228	-0.030747388087	0.1682260700…	
7	4.	0.168226070058	0.000013399715	0.1682272183…	
8	5.	0.168227218302	0.000000000003	0.1682272183…	
9	6.	0.168227218302	1.E-13	0.1682272183…	
10	7.	0.168227218302	0.	0.1682272183…	

Although this approach is very fast (in the previous examples only a few approximation steps were needed) it does not always converge, e.g. $x_0 = -2$ and $f(x) = 2 - Arc\tan(x)$. See graphs page.

	A st	B a	C fa	D	E
=					
1	step_i	x0	_	x_i+1	2.−12.*tan⁻¹(x)
2	_	_	_	_	
3	0.	-2.	15.2857846135	4.3690769223	
4	1.	4.3690769223	-14.1494883212	-19.31814891…	
5	2.	-19.3181489181	20.2289323372	611.47201313	
6	3.	611.47201313	-16.8299311651	-523779.7627…	
7	4.	-523779.762748	20.8495330111	476663654140.	
8	5.	476663654140.	-16.8495559215	-3.190298276…	
9	6.	-3.1902982765…	20.8495559215	1.7683903721…	
10	7.	1.76839037219…	-16.8495559215	-4.391000603…	
11	8.	-4.3910006036…	20.8495559215	3.3499826429…	
12	9.	3.34998264292…	-16.8495559215	-1.575768182…	
13	10.	-1.5757681821…	20.8495559215	4.3141994308…	
14	11.	4.31419943086…	-16.8495559215	-∞	
15	12.	-∞	20.8495559215	#UNDEF	
16	13.	#UNDEF	18.84955592153…	#UNDEF	



4.3 Newton's method – programming, a first attempt

We write a program **newton()** that requests the starting value a and the number of approximations to be done. The function f(x) is already defined in a calculation page.

```
newton                                          8/8
Define newton()=
Prgm
Local a,b,l,n
Request "starting value a = ",a
Request "number of approximations = ",n
For l,1,n
    b:=a-f(a)/(d/dx(f(x))|x=a)
    Disp "x",l," = ",b
    a:=b
EndFor
EndPrgm
```

```
f(x):=x^2-7                                    Done
newton()
 starting value a = 1
 number of approximations = 5
 x1. = 4.
 x2. = 2.875
 x3. = 2.65489130435
 x4. = 2.64576704419
 x5. = 2.64575131111
                                               Done
```

We can execute the program **newton()** in a calculation page or any other page of the same problem.

4.4 Newton's method – programming, an improved version

We write a program **newtonraphson()** that requests the starting value a and the tolerance levels for x and f(x).

- This version checks for every step of the process if $\left(\frac{d}{dx}(f(x))\,|\,x=a\right)\neq 0$
- The approximation process is terminated if at least one of the tolerance levels is met.
- No more than 30 steps are performed.
- As before, the function f(x) is already defined in a calculation page.

```
f(x):=x^3+x^2-x-1                                  Done
newtonraphson()
 starting value a = 5
 tolerance (for x) d = .0001
 tolerance (for f(x)) e = .0000001
 number of steps/approx = 7.
 root (approx) = 1.
                                                   Done
```

```
newtonraphson                                      7/18
Define newtonraphson()=
Prgm
Local a,b,d,e,zero,l
Request "starting value a = ",a
Request "tolerance (for x) d = ",d
Request "tolerance (for f(x)) e = ",e
l:=1
If (d/dx(f(x))|x=a)=0 Then
   Disp "Not converging – please change the starting value."
Else
   b:=a-f(a)/(d/dx(f(x))|x=a)
   While |b-a|≥d and |f(b)|≥e and (d/dx(f(x))|x=b)≠0 and l≤30
     a:=b
     b:=a-f(a)/(d/dx(f(x))|x=a)
     l:=l+1
   EndWhile
   zero:=b
   Disp "number of steps/approx = ",l-1
   Disp "root (approx) = ",zero
EndIf
```

```
f(x):=cos(x)-√7/7                                  Done
newtonraphson()
 starting value a = 0
 tolerance (for x) d = .0001
 tolerance (for f(x)) e = .0001
 Not converging – please change the starting value.
                                                   Done
```

```
f(x):=cos(x)-√7/7                                  Done
newtonraphson()
 starting value a = 1
 tolerance (for x) d = .0001
 tolerance (for f(x)) e = .0000001
 number of steps/approx = 2.
 root (approx) = 1.18319964021
                                                   Done
```

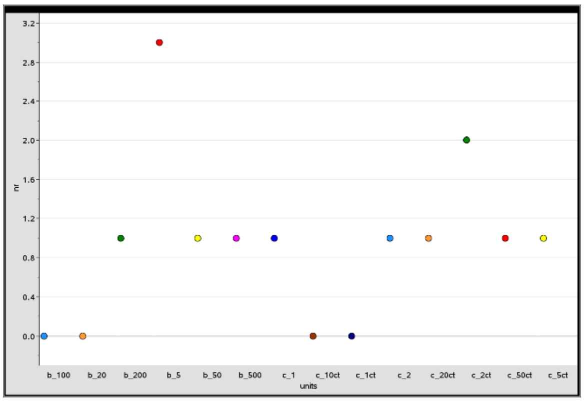
5. PROBLEM SOLVING – an example

TNS-file: 5.1 euro

Given a certain amount in €, how many banknotes and coins do you need to exactly match this amount? There are of course many answers possible unless you do it with as few notes and coins as possible.

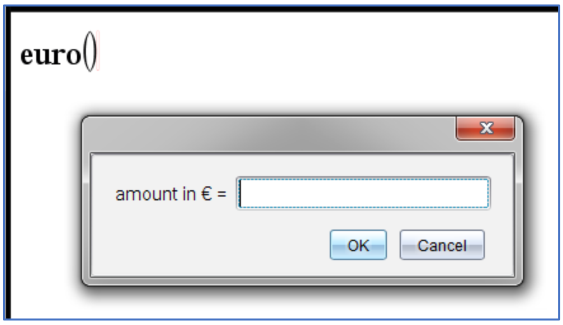
We can of course solve this problem with a spreadsheet. It is a nice opportunity to put the function **mod()** to work.

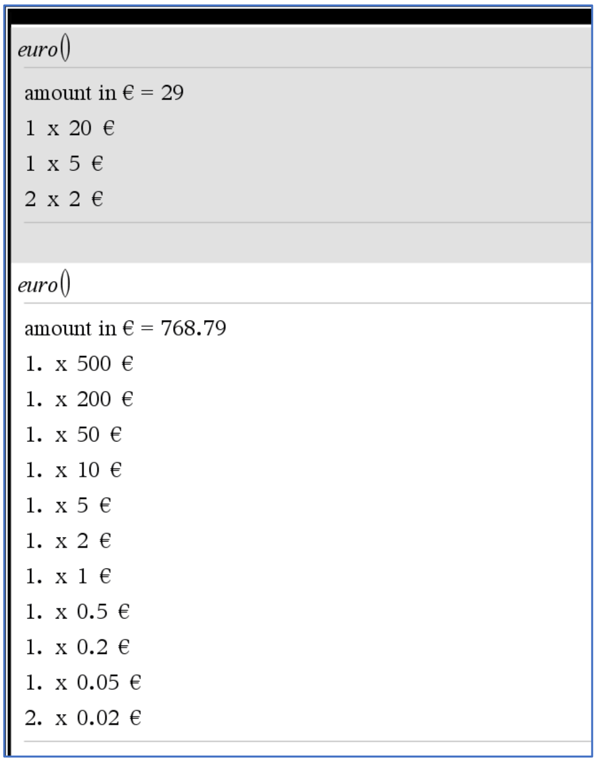
	A nr	B	C	D	E	F	G units
1	amount	768.79	euro				
2							
3	number	notes and coins			rest		notes_coins
4	1.	500			268.79		b_500
5	1.	200			68.79		b_200
6	0.	100			68.79		b_100
7	1.	50			18.79		b_50
8	0.	20			18.79		b_20
9	3.	5			3.79		b_5
10	1.	2			1.79		c_2
11	1.	1			0.79		c_1
12	1.	0.5			0.29		c_50ct
13	1.	0.2			0.09		c_20ct
14	0.	0.1			0.09		c_10ct
15	1.	0.05			0.04		c_5ct
16	2.	0.02			0.		c_2ct
17	0.	0.01			0.		c_1ct
18	6.	banknotes					
19	7.	coins					



An alternative method is to code a special programme also using the mod() function. Not only is this more efficient but it also allows us to create a more user friendly output.

```
* euro                                                    10/19
Define euro()=
Prgm
Local amount,rest,eu,quot
eu:={500,200,100,50,20,10,5,2,1,0.5,0.2,0.1,0.05,0.02,0.0}
Request "amount in € =",amount
For i,1,9
   rest:=mod(amount,eu[i])
   quot:=(amount-rest)/eu[i]
   If quot>0 Then
      Disp quot," x ",eu[i]," €"
   EndIf
   amount:=rest
EndFor
For i,10,15
   rest:=mod(amount·100,eu[i]·100)
   quot:=(amount·100-rest)/(eu[i]·100)
   If quot>0 Then
      Disp quot," x ",eu[i]," €"
   EndIf
   amount:=rest/100
EndFor
EndPrgm
```


euro()
amount in € =
OK
Cancel


euro()
amount in € = 29
1 x 20 €
1 x 5 €
2 x 2 €
euro()
amount in € = 768.79
1. x 500 €
1. x 200 €
1. x 50 €
1. x 10 €
1. x 5 €
1. x 2 €
1. x 1 €
1. x 0.5 €
1. x 0.2 €
1. x 0.05 €
2. x 0.02 €