Dr. Residual, part III

Rob.Corless@orcca.on.ca

This worksheet explores some of the ideas introduced in the last lecture.

```
> restart;
```

We work in "just" more than hardware floats, to ensure predictability (at the cost of speed)

```
> Digits := trunc(evalhf(Digits))+1;
```

$$Digits := 15$$

Now build the "four digit equally-spaced floating point numbers on [0,1)", and call it F4.

```
> F4 := [seq( Float(i,-4), i=0..9999 ) ]:
> nops( F4 );
```

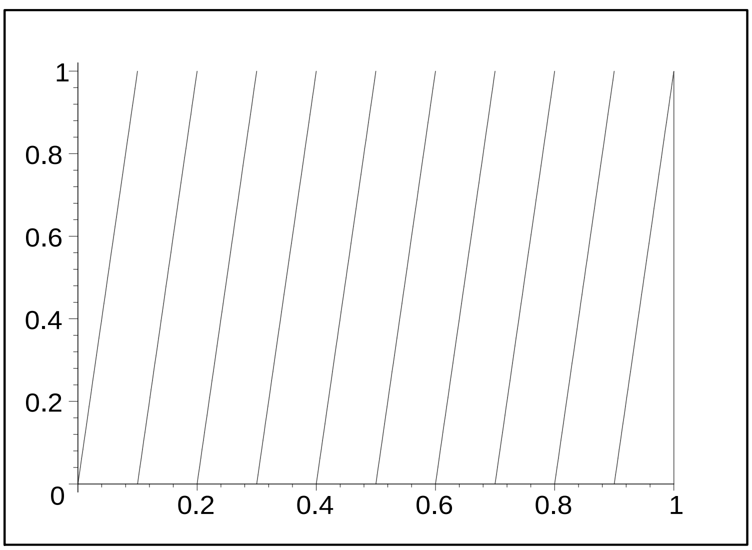
$$10000$$

Let's first look at the decimal shift map.

```
> bash := x -> frac( 10*x );
```

$$bash := x \to \mathrm{frac}(10\,x)$$

```
> plot( bash, 0..1, discont=true );
```




```
> F4[2];
```

$$0.0001$$

```
> bash( F4[2] );
```

$$0.0010$$

```
> once := map( bash, convert(F4,set) ):
> nops( once );
```

$$1000$$

```
> twice := map( bash, once ):
> nops( twice );
```

$$100$$

```
> thrice := map( bash, twice ):
> nops( thrice );
```

$$10$$

```
> thrice;
```

$$\{0., 0.1000, 0.2000, 0.3000, 0.4000, 0.5000, 0.6000, 0.7000, 0.8000, 0.9000\}$$

```
> last := map( bash, thrice );
```

$$last := \{0.\}$$

So we see that four iterations of "bash" take F4 to a single fixed point at 0. Not very interesting dynamics, and completely false to the dynamics on the interval [0,1). To see this, let's look at *exact* computation, which works in this case:

```
> x := 4/13;
```

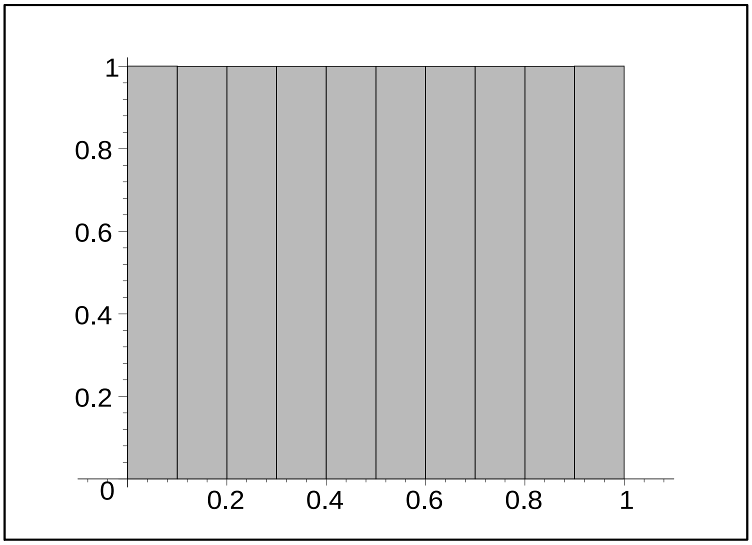
$$x := \frac{4}{13}$$

```
> seq( (bash@@i)(x), i=1..6 );
```

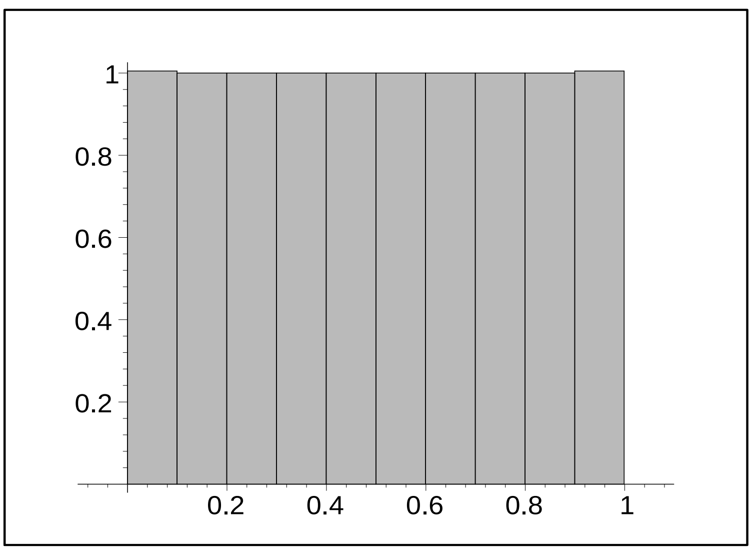
$$\frac{1}{13}, \frac{10}{13}, \frac{9}{13}, \frac{12}{13}, \frac{3}{13}, \frac{4}{13}$$

This shows that x = 4/13 has a periodic decimal expansion, as we know. But more can be shown, even in the floating-point case.

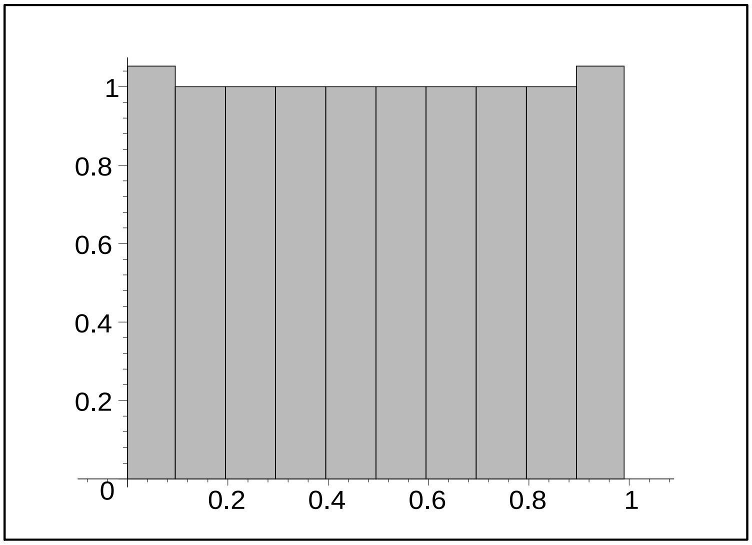
```
> ?histogram
> stats[statplots,histogram](convert(F4,list),numbars=10 );
```



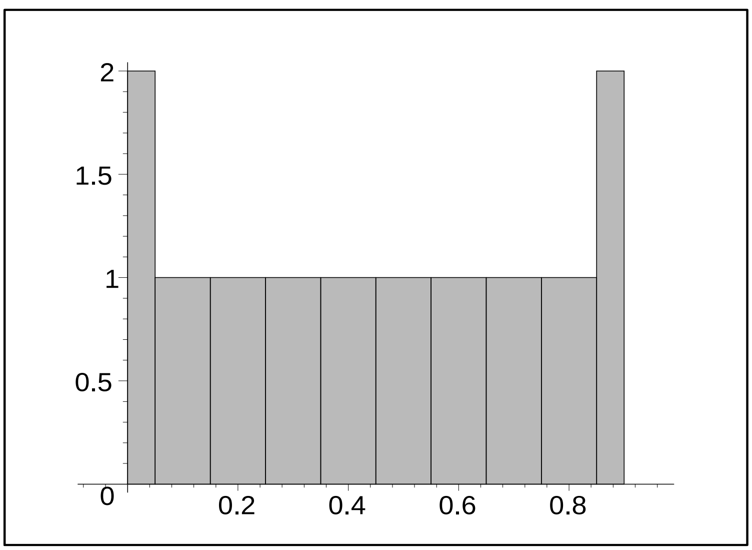

```
>  stats[statplots,histogram](convert(once,list),numbars=10 );
```




```
>  stats[statplots,histogram](convert(twice,list),numbars=10 );
```




```
> stats[statplots,histogram](convert(thrice,list),numbars=10 );
```



We see that the distribution remains, for a while, pretty good. One more iteration and it's completely wrong, of course.

Now let us look at a more interesting map, the Gauss map.

```
> Gauss := x -> `if`(x=0,0,frac( 1/x ));
```

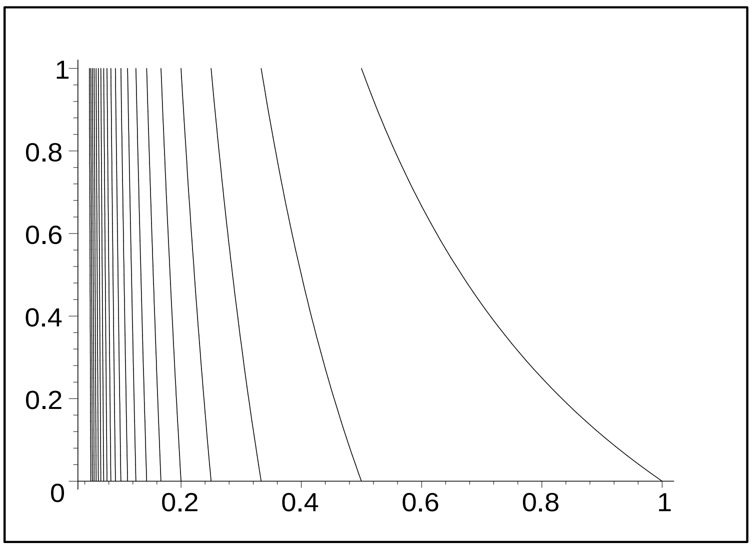
$$Gauss := x \to \text{`if`}(x = 0,\, 0,\, \text{frac}(\frac{1}{x}))$$

It's fairly hard to plot that nicely, but we find in Essential Maple a method to do it.

```
> x := 'x';
```

$$x := x$$

```
> plot( {seq( [1/(n+y),y,y=0..1], n=1..20)}, colour=black );
```



See the worksheet "Gauss.mws" for a plot of that on a torus. Now, let us investigate the Gauss map on the set F4.

```
> Digits := trunc(evalhf(Digits))+1;
```

$$Digits := 15$$

We need to embed the results of Maple's floating-point arithmetic into F4. One way is the following routine.

```
> embed := x -> Float( round( 10^4*x ), -4 );
```

$$embed := x \to \text{Float(round(10000*x),-4)}$$

```
> count := array(0..20):  count[0] := 10^4;
```

$$count_0 := 10000$$

```
> G1 := map( embed@Gauss, F4 ):
> count[1] := nops( convert(G1,set) );
```

$$count_1 := 6922$$

```
> G1[8];
```

$$0.5714$$

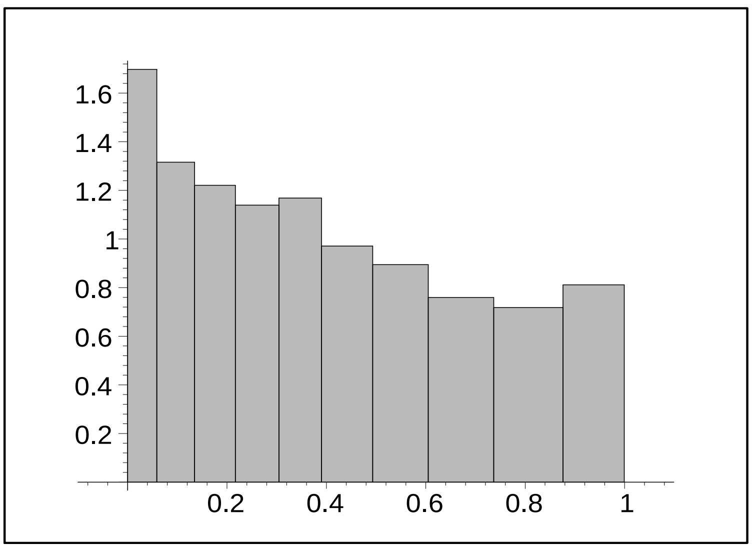
```
> G2 := map( embed@Gauss, convert(G1,set) ):
> count[2] := nops( G2 );
```

$$count_2 := 5075$$

```
> G3 := map( embed@Gauss, G2 ):
> count[3] := nops( G3 );
```

$$count_3 := 4056$$

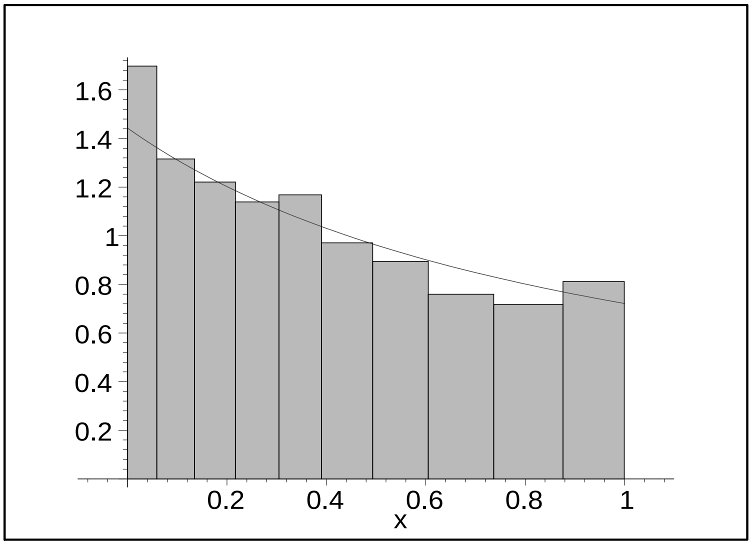
```
> for k from 4 to 20 do
> G||k := map( embed@Gauss, G||(k-1) ):
> count[k] := nops( G||k );
> end do:
> p1 := stats[statplots,histogram]( subs( undefined=missing, convert(
> G15, list )), numbars=10 ):  plots[display]( p1 );
```




```
> int( 1/(1+x), x=0..1 );
```

$$\ln(2)$$

```
> p2 := plot( 1/(1+x)/ln(2), x=0..1 ):  plots[display]( {p1,p2}
);
```




```
> with(networks):
> new(G):
> addvertex(F4,G):
> stink := {seq( evalf[1]( k/10 ) = Float(k*1000,-4), k=1..9 ),
seq(
> evalf[1](k/100)=Float(k*100,-4),k=1..9),
> seq(evalf[1](k/1000)=Float(k*1000,-4),k=1..9),seq(
> evalf[2](k/100)=Float(k*100,-4), k=1..99),
> seq(evalf[2](k/1000)=Float(k*10,-4),k=1..99),seq(
> evalf[3](k/1000)=Float(k*10,-4),k=1..999)}:
> stink[1];
```

$$0.818 = 0.8180$$

```
> evalf[2](9/100);
```

$$0.090$$

```
> edges := zip( (t1,t2)->[t1,t2], F4, G1 ):
> edges[1..20];
```

$$[[0., 0.], [0.0001, 0.], [0.0002, 0.], [0.0003, 0.3333], [0.0004, 0.], [0.0005, 0.], [0.0006, 0.6667], [0.0007, 0.5714], [0.0008, 0.], [0.0009, 0.1111], [0.0010, 0.], [0.0011, 0.0909], [0.0012, 0.3333], [0.0013, 0.2308], [0.0014, 0.2857], [0.0015, 0.6667], [0.0016, 0.], [0.0017, 0.2353], [0.0018, 0.5556], [0.0019, 0.3158]]$$

```
> st := time():
> for i to nops(edges) do addedge( edges[i], G ): end do:
> time() - st;
```

$$79.695$$

```
> st := time():  comps := components(G): time() - st;
```

$$112.181$$

```
> nops( comps );
```

$$16$$

```
> sizes := map( nops, comps );
```

$$sizes := \{6, 7, 13, 22, 37, 38, 39, 43, 64, 71, 95,\\ 201, 273, 1422, 1476, 6193\}$$

```
> smallest := min( op(sizes) );
```

$$smallest := 6$$

```
> biggest := max( op(sizes) );
```

$$biggest := 6193$$

```
> small := select( t->nops(t)=smallest, comps )[1];
```

$$small := \{0.0011, 0.0909, 0.1100, 0.9009, 0.9167,\\ 0.9989\}$$

```
> Gauss(0.0011);
```

$$0.090909090909$$

```
> embed(%);
```

$$0.0909$$

```
> Gauss(%);
```

$$0.0011001100110$$

```
> embed(%);
```

$$0.0011$$

```
> map( embed@Gauss, small );
```

$$\{0.0011, 0.0909, 0.1100\}$$

```
> map( embed@Gauss, % );
```

$$\{0.0011, 0.0909\}$$

```
> big := select( t->nops(t)=biggest, comps )[1]:
> fixed := false;
```

$$fixed := false$$

```
> nxt := map( embed@Gauss, big ):
> nops( big );
```

$$6193$$

```
> nops( nxt );
```

$$4296$$

```
> tmp := map( embed@Gauss, nxt ):
> fixed := evalb( nops( nxt ) = nops ( tmp ) );
```

$$fixed := false$$

```
> for k while not fixed do
> nxt := tmp;
> tmp := map( embed@Gauss, nxt ):
> fixed := evalb( nops(nxt) = nops( tmp ) ) ;
> end do:
> k;
```

$$109$$

```
> nops( tmp );
```

$$1$$

```
> convert(tmp,list);
```

$$[0.]$$

```
> map( embed@Gauss, % );
```

$$[0.]$$

```
> ?fibonacci
> seq( combinat[fibonacci](k)/combinat[fibonacci](k+1), k=1..15
);
```

$$1, \frac{1}{2}, \frac{2}{3}, \frac{3}{5}, \frac{5}{8}, \frac{8}{13}, \frac{13}{21}, \frac{21}{34}, \frac{34}{55}, \frac{55}{89}, \frac{89}{144}, \frac{144}{233}, \frac{233}{377}, \frac{377}{610}, \frac{610}{987}$$

```
> map( t->denom(t)^2, [%] );
```

$$[1, 4, 9, 25, 64, 169, 441, 1156, 3025, 7921, 20736, 54289, 142129, 372100, 974169]$$

```
> evalf( (-1+sqrt(5))/2 );
```

$$0.618033988749900$$

```
> embed(%);
```

$$0.6180$$

```
> (embed@Gauss)(%);
```

$$0.6181$$

```
> (embed@Gauss)(%);
```

$$0.6179$$

```
> (embed@Gauss)(%);
```

$$0.6184$$

```
> (embed@Gauss)(%);
```

$$0.6171$$

```
> (embed@Gauss)(%);
```

$$0.6205$$

```
> (embed@Gauss)(%);
```

$$0.6116$$

```
> (embed@Gauss)(%);
```

$$0.6351$$

```
> (embed@Gauss)(%);
```

$$0.5746$$

```
> (embed@Gauss)(%);
```

$$0.7403$$

```
> (embed@Gauss)(%);
```

$$0.3508$$

```
> (embed@Gauss)(%);
```

$$0.8506$$

```
> (embed@Gauss)(%);
```

$$0.1756$$

```
> F4[6219];
```

$$0.6218$$

```
> hmm := convert(%,confrac);
```

$$hmm := [0, 1, 1, 1, 1, 1, 1, 4, 3, 1, 7, 4]$$

```
> cval := proc( L::list ) local i, s, n;
> n := nops( L );
> s := L[n];
> for i from n-1 by -1 to 1 do
> s := L[i] + 1/s;
> end do
> end proc:
> cval (hmm[1..8] );
```

$$\frac{74}{119}$$

```
> evalf[4]( % );
```

$$0.6218$$