Physics 651: Assignment 1

(to be submitted by Thursday, August 31, 2023)

1. You can think of the finite field $\mathbb{F}_n$ as the ring of integers $\{0, 1, 2, \dots, n-1\}$, defined modulo n.

   (a) Generate lists of the elements in $\mathbb{F}_5$ in *Mathematica* using `Range` and `Table`.

```
Range[0, 4]
Table[i, {i, 0, 4}]
```

   Note that pressing enter or return on your keyboard will produce just a bare line-feed/carriage-return. To get *Mathematica* to interpret and execute a line of code, you will have to type shift + return. Next, generate the set $A = \{1/(1+k^2) : k \in \mathbb{F}_5\}$. Here are four ways to do so:

```
A = 1/(1 + Range[0, 4]^2)
A = Map[Function[1/(1 + #^2)], Range[0, 4]]
A = Map[1/(1 + #^2) &, Range[0, 4]]
A = Array[1/(1 + (# - 1)^2) &, 5]
```

   In the second and third lines above, `Map` applies the anonymous function $k \to 1/(1+k^2)$—expressed in code as `1/(1+#^2)` and wrapped in a `Function` call or terminated with the `&` symbol—to each element of the list.

   Write your own code to compute A, based on a call to `Table`.

   (b) Determine the addition and multiplication tables for $\mathbb{F}_3$. Compute all the elements by hand.

   (c) Verify that your calculation agrees with the two tables produced algorithmically by the following *Mathematica* code:

```
Table[Table[Mod[i+j, 3], {i, 0, 2}], {j, 0, 2}] // MatrixForm
Table[Table[Mod[i*j, 3], {i, 0, 2}], {j, 0, 2}] // MatrixForm
```

   As a convenience, the `Table` function supports a simplified syntax for nested tables.

```
Table[Mod[i+j, 3], {i, 0, 2}, {j, 0, 2}] // MatrixForm
Table[Mod[i*j, 3], {i, 0, 2}, {j, 0, 2}] // MatrixForm
```

   Also try this with the `// MatrixForm` specification removed, which lets you see the unformatted result. Observe that the output is a nested list of lists. This is the data structure that *Mathematica* uses for matrices and multidimensional arrays.

   Alternatively, one can map an anonymous function that handles the modulo arithmetic onto the outer product (`Outer`) of the lists produced by `Range`.

```
Outer[Function[Mod[#1+#2, 3]], Range[0, 2], Range[0, 2]] // MatrixForm
Outer[Mod[#1*#2, 3] &, Range[0, 2], Range[0, 2]] // MatrixForm
```

   (d) Modify the codes from question 1c so that they produce the formatted addition and multiplication tables of $\mathbb{F}_4$.

   (e) Consider a triply nested `Table` structure representing all possible sums of three numbers in $\mathbb{F}_5$. We can then confirm that the set of all possible values covers all of $\mathbb{F}_5$.

```
T = Table[Mod[i+j+k, 5], {i, 0, 4}, {j, 0, 4}, {k, 0, 4}]
Flatten[T]
Union[Flatten[T]]
DeleteDuplicates[Flatten[T]] (* This is equivalent to the previous line *)
Range[0,4] == %
```

Explain what is going on in the last four lines of this code snippet. What does `Flatten` do? What purpose do `Union` and `DeleteDuplicates` serve? What is the meaning of the `%` symbol?

Finally, rewrite the code using `Outer` and `Range`.

(f) Create a quadruply nested list structure representing all possible *products* of four numbers in $\mathbb{F}_{10}$. Verify that the set of values covers $\mathbb{F}_{10}$.

2. *Mathematica* offers a variety of input methods for special characters and symbols. Most characters have a full name (`\[FullName]`) and an associated alias (esc `alias` esc) for faster input. For example, ∞, π, and e are produced by `\[Infinity]` `\[Pi]`, and `\[ExponentialE]` but also by esc `inf` esc, esc `pi` esc, and esc `ee` esc. Some characters also have a standalone name that doesn't require an escape sequence: `Infinity` and `E` can be used unadorned. Run the following code to establish membership of various numbers and expressions in the integers ($\mathbb{Z}$), reals ($\mathbb{R}$), and rationals ($\mathbb{Q}$).

```
Element[-256, Integers]
Element[11/32, Integers]
Element[\[Infinity], Reals]
0 esc el esc NonNegativeReals
Element[\[Pi], Reals]
esc ee esc  esc el esc Rationals
Element[-2/3, Rationals]
2/3 \[Element] Reals \[And] 2/3 \[Element] Rationals
Element[52, Reals] esc and esc Element[52, Integers]
(* The Sinc function has a well-defined real-valued limit at zero. *)
Element[Limit[Sin[x]/x, x -> 0],Reals]
(* Every individual element of this summation is rational, but the total sum is real. *)
Apply[And, Table[1/n^2 \[Element] Rationals, {n, 1, 100}]]
Sum[1/n^2, {n, 1, Infinity}] esc el esc Reals
```

As a *Mathematica* coding exercise, show that $\sum_{n=0}^{\infty}(-1)^n/n! = 1/e$ is real but that the first 101 terms of the series are individually rational.

3. *Mathematica* supports complex numbers via an imaginary $i = \sqrt{-1}$, represented by `\[ImaginaryI]`, esc `ii` esc, or just `I`. Hence, `1 + 2I` and `2 - 3*I` are valid numbers in $\mathbb{C}$ (`Complexes`). Multiplication is inferred, so products can be expressed as either `a` space `b` or `a*b`. In the following code, note the difference between the *assignment* operator (`=`) and the *test-for-equality* operator (`==`). Further observe that *logical and* has both an operator (`&&`) and function form (`And`). Take care to distinguish the semantics of a `Symbol` and `String`. The latter supports concatenation (via the function `StringJoin` or operator `<>`).

```
(* What follows are four ways to define the same complex number *)
a1 = Complex[1,2]
a2 = 1+2\[ImaginaryI]
a3 = 1+2 esc ii esc
a = 1+2I
(* Here we verify that they are equal*)
EqualTo[a1][a]
a1 == a && a2 == a
a1 == a esc and esc a2 == a esc and esc a3 == a
Apply[And,Table[Symbol["a" <> ToString[i]] == a , {i, 1, 3}]]
(* The complex numbers are closed under addition, subtraction, multiplication, and
    division*)
b = 2 + 3I
z1 = a+b
z2 = a-b
z3 = a b
z4 = a/b
z1 esc el esc Complexes
```

```
z2 esc el esc Complexes
Apply[And,Table[Symbol["z" <> ToString[i]] \[Element] Complexes , {i, 1, 4}]]
Norm[z1]
N[Norm[z1]]
Arg[z1]
N[%, 20]
Norm[z2]
Arg[z2]
For[k = 1, k < 5, ++k, zklabel = "z" <> ToString[k]; zk = Symbol[zklabel];
 Print[zklabel, " = ", zk, ": ", "\nnorm(", zklabel, ") = ", Norm[zk],
  ",\narg(", zklabel, ") = ", Arg[zk]]]
```

Explicit numerical evaluation is carried out with the function `N`, which takes an optional argument that specifies the precision (the default is 8 decimal digits). The `For` loop uses a C-like *initialize–test–increment–body* syntax. `Print` outputs text and mathematical expressions to the screen; it is aware of standard string escape sequences, such as `\t` (tab) and `\n` (newline).

Write a program that defines complex numbers $z_k = \cos(\pi k/5) + i\sin(\pi k/5)$ for $k = 0, 1, \ldots, 9$ and for each k verifies that $|z_k| = 1$, $\operatorname{Arg} z_k = \pi k/5 \pmod{2\pi}$, and $\exp(i \operatorname{Arg} z_k) \doteq z_k$. Where $\doteq$ is indicated, establish that the expressions on the left- and right-hand side agree to within 12 digits.

4. The value $\phi = (1+\sqrt{5})/2$ can be computed with the expression `(1+Sqrt[5])/2` or directly invoked with the special named constant `GoldenRatio`. ϕ is unique in that it has an infinite continued fraction representation in which all the terms are 1:

$$\phi = 1 + \frac{1}{1 + \frac{1}{1+\frac{1}{1+\cdots}}}$$

Mathematica has some useful helper functions for continued fractions. `ContinuedFraction` converts a number or mathematical expression into a (truncated) list of its coefficients in the continued fraction representation. Conversely, `FromContinuedFraction` takes the finite list of coefficients and evaluates the corresponding number as a partial continued fraction. It's instructive to observe how the partial continued fractions converge to ϕ as the length of the coefficient list grows. Take time to be sure that you understand the `ReplaceAll` (`/.`) trick that allows us to compute the terms explicitly.

```
ContinuedFraction[GoldenRatio, 1]
FromContinuedFraction[%]
1
ContinuedFraction[GoldenRatio, 2]
FromContinuedFraction[%]
1+1
ContinuedFraction[GoldenRatio, 3]
FromContinuedFraction[%]
1+1/(1+1)
ContinuedFraction[GoldenRatio, 4]
FromContinuedFraction[%]
1+1/(1+1/(1+1))
ContinuedFraction[GoldenRatio, 5]
FromContinuedFraction[%]
1+1/(1+1/(1+1/(1+1)))
expr = 1 + w; Do[expr = expr /. w -> 1/(1 + w); Print[N[expr /. w -> 0, 12]], 20]
ListLinePlot[Table[{k, FromContinuedFraction[ContinuedFraction[GoldenRatio, k]]}, {k, 1,
   10}], PlotMarkers -> Automatic, PlotRange -> {{0, 11}, {0.5, 2.5}}, Frame -> True]
```

Now consider the sequence of partial continued fractions,

$$\pi_1 = 3,\ \ \pi_2 = 3 + \frac{1}{7},\ \ \pi_3 = 3 + \cfrac{1}{7 + \cfrac{1}{15}},\ \ \pi_4 = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \frac{1}{1}}},\ \ \pi_5 = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \frac{1}{1 + \frac{1}{292}}}},\ \ldots$$

Each π_k is the continued fraction that has been truncated at order k. These achieve the asymptotic value $\lim_{k\to\infty} \pi_k = \pi$; the sequence approaches its limit from below. Investigate the convergence by making a table of the first 15 values $(\pi_1, \pi_2, \ldots, \pi_{15})$. Then make a plot (with a logarithmic vertical axis, using `ListLogPlot`) of $\pi - \pi_k$ for k ranging from 1 to 50.

5. *Mathematica* supports various `Function` styles and other syntactic sugar (`UseShorthandNotations`). The hash (`#`) represents the single argument passed to a unary function; `#1`, `#2`, ... represent the various arguments passed to a multivariable function. The ampersand (`&`) is the terminator for an anonymous function definition. The operators `/@` and `@@@` are stand-ins for `Map` and `MapApply`. List-element selection and manipulation (`ElementsOfLists`) can be carried out with the functions `Take`, `Drop`, `Part`, and `Span` but also with a double-square-bracket indexing-and-slicing notation (`[[;;]]`).

```
(* c1 is a list of complex numbers; c2 and c3 are identity transformations of c1 *)
c1 = 1 + Range[5] I
c2 = Map[Norm[#] Exp[I Arg[#]] &, c1]
c3 = Norm[#] Exp[I Arg[#]] & /@ c1
N[c1]
N[c2]
c3 // N
Table[Simplify[c1[[i]] == c2[[i]]], {i, 0, 5}]
Simplify[MapThread[#1==#2&,{c1,c2}]]
Simplify[Thread[c1==c2]]
Simplify[Equal @@@ Table[{c1[[i]], c2[[i]]}, {i, 1, Length[c1]}]]
(* Plot spirals in the complex plane *)
F[a_, n_] := (1 + a Range[0, n]/(n/2)) Exp[(\[Pi]/(2 n/5)) I Range[0, n]];
ComplexListPlot[{F[2, 200], F[3, 200], F[4, 200]}, PlotLegends -> {"a=2", "a=3", "a=4"}]
(* There are several ways to index and slice lists *)
u = Fibonacci /@ Range[15]
u[[1]]
u[[-1]] (* This is a python-style syntax that selects the last element of the list *)
u[[9]]
Part[u,9]
u[[3 ;; 7]]
Take[u, {3, 7}]
u[[;; 5]]
Take[u, 5]
u[[-5 ;;]]
Take[u, -5]
```

The famous Fibonacci sequence $(f_n) = (f_1, f_2, f_3, \ldots) = (1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots)$ is defined recursively by $f_0 = f_1 = 1$ and $f_{n+1} = f_n + f_{n-1}$. Consider a sequence transformation $(f_n \to g_n = (f_{3n-2} f_{3n-1} f_{3n})^{1/3})$ that maps three consecutive Fibonacci values into their geometric average:

$$(g_n) = \left((f_1 f_2 f_3)^{1/3},\ (f_4 f_5 f_6)^{1/3},\ (f_7 f_8 f_9)^{1/3},\ \ldots\right).$$

Using *Mathematica*, populate a list with the first 10 elements of (g_n). As a challenge, see if you can accomplish this with a program consisting of only one line of code. The list elements should be exact symbolic results. Compare them to the following decimal approximation to confirm their correctness:

$$(g_n) \doteq (1.25992, 4.93242, 21.0159, 88.9963, 377.001, \ldots).$$

Finally, use `Show` to superimpose a `ListLogPlot` of g_n and a `LogPlot` of the heuristic $0.29e^{1.44n}$.