Computing for Mathematical Physics 2022/23

Homework7

Mark for homework7: /33

(to be competed by your marker)

Feedback from marker:

(to be competed by your marker)

Give your answers in the code cells marked (* Enter your solution here *)

Double click the vertical braces on the RHS of each of the question headings to open and view them.

- 1. Generating and summing (powers of) list elements. [5 marks]
  - a) Define a function `plusSquare[a_, x_]`, which returns $a + x^2$. [1 mark]

In[1]:=
```
plusSquare[a_, x_] := a + x^2
```

  - b) Define a function `sumOfSquaresV1[list_]`, calling `plusSquare` inside `Fold`, returning the sum of the squares of each element inside `list`. Display the output of `sumOfSquaresV1[Range[10]]`. [2 marks]

```
(*retrieve function and put in fold,
result will add the squares of each number*)
sumOfSquaresV1[list_] := Fold[plusSquare, list]
```

In[3]:=
```
sumOfSquaresV1[Range[10]]
```

- c) Define a function `sumOfSquaresV2[list_]`, which is the same as `sumOfSquaresV1` except that it does not call `plusSquare`, but instead uses a pure function, defined in terms of #'s, to achieve the same output. Display the output of `sumOfSquaresV2[Range[10]]`. [2 marks]

```
(*pure function uses the same structure as the defined function*)
sumOfSquaresV2[list_] := Fold[#1 + #2^2 &, list]
```

```
In[5]:= sumOfSquaresV2[Range[10]]
```

- 2. Simulating dice rolls with `FoldList`. [9 marks]

- a) Write a function, `rollDice[n_]`, simulating `n` rolls of a *biased* six-sided dice, by returning `n` random integers, between 1 and 6 [inclusive], using `RandomChoice`. The implementation of `RandomChoice` should reflect that the probability to roll a six with the biased six-sided dice is $\frac{3}{8}$ whilst the probability of rolling any other value is $\frac{1}{8}$.
[2 mark]

```
In[1]:= (*Give bias towards 6*)
rollDice[n_] :=
 RandomChoice[{0.125, 0.125, 0.125, 0.125, 0.125, 0.375} → {1, 2, 3, 4, 5, 6}, n]
```

- b) Using `FoldList`, define a function `cumulative[inputList_]`, returning a list which gives the cumulative sum of the elements in `inputList`; i.e. the *i*'th element in the returned list is given by the sum of all elements up to and including the *i*'th element in the `inputList`. E.g. `cumulative[{p,q,r,s}]` should return `{p, p+q, p+q+r, p+q+r+s}`. [1 mark]

```
In[2]:= (*add up all elements*)
cumalative[inputList_] := FoldList[Plus, inputList]
```

- c) Set `nRolls=100000`. Create a list, `simulatedRolls`, of `nRolls` dice rolls using `rollDice`. Create a further list `simulatedCumulative` , from `simulatedRolls` and the `cumulative` function. ***Suppress*** the on-screen display of `simulatedRolls` and `simulatedCumulative` by terminating each of the commands which generate them with a semi-colon. [1 mark]

```
In[3]:= nRolls = 100000;
```

```
In[4]:= simulatedRolls = rollDice[nRolls];
simulatedCumalative = cumalative[simulatedRolls];
```

- d) The **expected** cumulative total for a *fair* dice would be the the expectation, i.e. 3.5, times the number of rolls. Generate a list, `expectedCumulative`, using `Range`, with `nRolls` elements, where the *i*'th element equals $\mathcal{E}\, i$, where $\mathcal{E}$ is the expectation for our *biased* dice. [2 marks]

Expectation value: 1+2+3+4+5+6+6+6 / 8 = 33/8 = 4.125

```
In[6]:= expectedCumulative = Map[Times[4.125, #] &, Range[nRolls]];
```

- e) Plot `simulatedCumulative` and `expectedCumulative` on the same graph. Make a further, separate, plot of the ratio `simulatedCumulative/expectedCumulative`, setting the y-axis range to {0.98,1.02}. [3 marks]

In[40]:=
```
ListLinePlot[{simulatedCumalative, expectedCumulative},
 (*Plot both using listlineplot which automatically joins them together*)
 PlotLegends → {"Simulated", "Expected"}, PlotLabel → "Dice rolls",
 AxesLabel → {"no. dice rolls", "cumalative sum"}
 (*Label axis, graphs and add legends*)
]
```

In[43]:=
```
ListLinePlot[sim / exp, (*Plot the ratio between
  the two cumaltives by dividing them by each other*)
 PlotLabel → "Ratio of simulated over expected cumalatives",
 AxesLabel → {"no. dice rolls", "Ratio of cumalatives"},
 (*Label axis, graph again etc.*)
 PlotRange → {0.98, 1.02} (*add plotrange*)
]
```

- 3. Hypergeometric function with `Fold` and `Nest`. [9 marks]
  - The *Pochhammer* symbol $(q)_n$ is defined by:

    - eq. 1 : $(q)_n = \begin{cases} 1 & n = 0\,, \\ q(q+1)\ldots(q+n-1) & n > 0\,. \end{cases}$

  - a) Write a function, `pochhammerUsingFold[q_,n_]`, using `Fold`, returning the Pochhammer symbol $(q)_n$, as defined in eq. 1.

    - Display the result of `pochhammerUsingFold[q,0]`

    - Using `Simplify` *and* the `==` operator, test that `pochhammerUsingFold[q,nTerms]`, yields an identical result to *Mathematica*'s internal `Pochhammer[q,nTerms]` function, for `nTerms=7` and `nTerms=18.` [4 marks]

Pochhammer expansion: q(q+1)(q+2)(q+3)....(q+(n-1))

In[45]:=
```
pochhammerUsingFold[q_, n_] := If[n == 0, 1, Fold[(#1 ) (q + #2) &,
   (*get the previous result and multiply it by q + (n-1)*)
   q, Range[n - 1]]]
pochhammerUsingFold[q, 7] == Pochhammer[q, 7]
pochhammerUsingFold[q, 18] == Pochhammer[q, 18]
```

  - b) Replace the line "`MISSING CODE`" inside the `pochhammerUsingNestList[q_,nMax_]` function, defined below, such that it returns a list of all Pochhammer symbols $(q)_n$, from $n = 0$ up to and including

n = nMax (see code comments for more details). Run your function to produce output for `pochammerUsingNestList[q,4]`.
[2 marks]

In[37]:=

```
pochammerUsingNestList[q_, nMax_] :=
  NestList[
   (* The following (pure) function returns a two-element list *)
   (* every time it is called.                                   *)
   (*  • The 1st element is the 1st element in the previous     *)
   (*    two-element list that it computed, #[[1]], plus 1,     *)
   (*    starting with 1; i.e. this element goes 1, 2, 3, ...   *)
   (*  • The 2nd element is always "MISSING CODE".               *)
   (*    You must swap "MISSING CODE" for a combination of      *)
   (*    q, #[[1]], #[[2]], s.t. on the 0th iteration this      *)
   (*    element is 1, on the 1st it's q, on the 2nd it's       *)
   (*    q(q+1), on the 3rd it's q(q+1)(q+2) etc ...             *)
   {
    #[[1]] + 1,
    ,
    (* Initial 2-element list, output on the 0th iteration is {0,1} *)
    {0, 1},
    (* Do nMax iterations after the 0th iteration. *)
    nMax
    ]
```

In[53]:=

```
pochammerUsingNestList[q_, nMax_] := NestList[{#[[1]] + 1,
     (#[[2]] ) (q + #[[1]])} &, (*follows a very similair pattern
    to the question before except adapted to how nest works*)
   {0, 1}, nMax]
pochammerUsingNestList[q, 4][[1 ;;, 2]]
```

In[56]:=

```
{{1, 10}, {2, 20}, {3, 30}, {40, 40}}[[1 ;; 4, 2]]
(*some experimentation for the command before,
normally i remove these but i thought it would
 be nice to see how i test code to get to my results *)
```

- c) The hypergeometric function ${}_2F_1(a, b, c, z)$ is defined for $|z| < 1$ by the power series

- eq. 2 : ${}_2F_1(a, b, c, z) = \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{(c)_n} \frac{z^n}{n!}$

where $(a)_n$, $(b)_n$, $(c)_n$ are all Pochhammer symbols.

i) The function, `hypergeometricExpansion[a_,b_,c_,z_,nMax_]`, written in the code cell below, is intended to return the RHS of eq. 2 up to and *including* terms $\propto z^{\text{nMax}}$. The latter function depends on a second function `term[a_,b_,c_,z_,n_]` which we have left undefined. *Define* `term[a_,b_,c_,z_,n_]`, below, such that `hypergeometricExpansion` will then function as intended.

• Should you need Pochhammer symbols, use *Mathematica*'s native implementation of these: see `??Pochhammer`.

ii) Use `Series` to expand *Mathematica*'s native implementation of ${}_2F_1(a, b, c, z)$, called `Hypergeometric2F1[a,b,c,z]` , about `z=0`, for terms up to and *including* $\propto z^5$ [i.e. the output of `Series` is expected to end with $O(z^6)$]. Convert the output from a `SeriesData` object to a normal expression. Store this result as `native2F1Expansion`.

iii) Use `==` and `Simplify` to test whether `native2F1Expansion` equals the output from your `hypergeometricExpansion[a,b,c,z,nMax]` function for `nMax=5`.

[3 marks]

```
hypergeometricExpansion[a_, b_, c_, z_, nMax_] :=
 1 +
  Fold[
   (#1 + term[a, b, c, z, #2]) &,
   0,
   Range[nMax]
  ]
(* Define the RHS of fn, term[n_], below, s.t. *)
(* hypergeometricExpansion[a,b,c,z,nMax] will  *)
(*return all terms on the RHS of eq. 2, above, *)
(* up to and including O(z^nMax) *)
term[a_, b_, c_, z_, n_] :=
 (Pochhammer[a, n] × Pochhammer[b, n])/Pochhammer[c, n] z^n/n! (*substitution of Pochhammer
   functions and "z" and "n" variables is extremely straightforward*)

native2F1Expansion = Normal[Series[Hypergeometric2F1[a, b, c, z], {z, 0, 5}]];
hypergeometricExpansion[a, b, c, z, 5] ==
 native2F1Expansion (*Simplify function was not required :)*)
```

■ 4. Monte Carlo integration with `NestWhile`. [10 marks]

■ In the input cell below the function $g[x, y]$ is defined as

•

$$g[x,y] = \mathrm{Cos}\left[\tfrac{1}{2}\sqrt{x^2+y^2}\right]^2 \mathrm{Cos}[2\,\mathrm{ArcTan}[x,y]]^2\,\Theta\left(3\pi-\sqrt{x^2+y^2}\right)\Theta\left(\sqrt{x^2+y^2}-\pi\right).$$

A 3D surface plot is also provided to show the form of the function.

In this question we seek to perform the following integral
by so-called Monte Carlo methods

- $\mathcal{I} = \int_{y_{min}}^{y_{max}} \int_{x_{min}}^{x_{max}} g[x, y]\, dx\, dy,$

with $\{x_{min}, x_{max}\} = \{y_{min}, y_{max}\} = \{-3\pi, 3\pi\}$.

In[1]:=
```
(* Your answer to this question should use  *)
(* the following xMin, xMax, yMin, yMax and *)
(* g[x_,y_] definitions. *)
{xMin, xMax} = {yMin, yMax} = {-3 π, 3 π};

g[x_, y_] :=
 Cos[1/2 Sqrt[x^2 + y^2]]^2 Cos[2 ArcTan[x, y]]^2
  UnitStep[3 π - Sqrt[x^2 + y^2]] × UnitStep[Sqrt[x^2 + y^2] - π]

Plot3D[
 (* Function to plot *)
 g[x, y],

 (* Variables and ranges to plot w.r.t *)
 {x, xMin, xMax},
 {y, yMin, yMax},

 (* Presentation *)
 PlotRange → All,
 PlotPoints → 100,
 AxesLabel → {"x", "y", "z"},
 LabelStyle → {Black, Bold, FontSize → 14},
 ColorFunction → "TemperatureMap",
 PlotLabel → "Surface plot of g[x,y]"
]
```

- a) Determine the integral $\mathcal{I}$ numerically with `NIntegrate[...]`.
  [1 mark]

In[4]:=
```
NIntegrate[g[x, y], {x, xMin, xMax}, {y, yMin, yMax}]
```

- b) Define a function `onePointApproximation[]`, with *no* inputs, returning $g[x_{R_1}, y_{R_2}] * (x_{max} - x_{min}) * (y_{max} - y_{min})$, where x_{R_1} and y_{R_2} are two *independent real* random numbers in $x_{min} \le x_{R_1} \le x_{max}$ and $y_{min} \le y_{R_2} \le y_{max}$ respectively.
  [2 marks]

In[5]:=
```
onePointApproximation[] :=
  g[RandomReal[{xMin, xMax}], RandomReal[{yMin, yMax}]] *
   (xMax - xMin) * (yMax - yMin)
```

- c) *Read* the following description, *study* the template `updateIntegration[...]` `Module` code in the input cell below, and *insert* code for the four missing elements inside it that are marked by comments.

  The module's input is a four-element list comprised of:

  - `nPoints`:
    number of times `onePointApproximation[]` has been called in the course of the current integration; this should be the same as the number of times the `Module[...]` has been called.
  - `avg`:
    average of all values returned by `onePointApproximation[]`, which is an approximation for the integral $\mathcal{I}$.
  - `err`:
    the uncertainty on `avg`; you may assume that `nPoints` is large if need be.
  - `avgOfSquares`:
    average of the [`nPoints`] *squares* of values returned by `onePointApproximation[]`.

  The job of the module is to call `onePointApproximation[]` to generate another one-point approximation to the integral, update all four quantities above based on the latter call, and return them *in the same form* as the initial four-element input list.
  [4 marks]

```
(* Complete Module below: in particular replace  *)
(* the magenta comments by the appropriate code. *)
```

In[6]:=
```
updateIntegration[{nPoints_, avg_, err_, avgOfSquares_}] :=
 Module[{
   nextNPoints,
   nextOnePointApproximation,
   nextAvg,
   nextErr,
   nextAvgOfSquares
  },

  nextNPoints =
   nPoints + 1;

  nextOnePointApproximation =
   onePointApproximation[];

  nextAvg =
   1/nextNPoints * (avg * nPoints + nextOnePointApproximation);

  nextAvgOfSquares =
   1/nextNPoints * (avgOfSquares * nPoints + nextOnePointApproximation^2);

  nextErr =
   (1 / Sqrt[nextNPoints]) * Sqrt[nextAvgOfSquares - nextAvg^2];

  {nextNPoints, nextAvg, nextErr, nextAvgOfSquares}
 ]
```

- d) Using `NestWhile` with `updateIntegration[...]` perform a numerical evaluation of the integral $\mathcal{I}$. Specifically, `NestWhile` should call `updateIntegration[...]` repeatedly as long as `err` ≥ 0.2 or `nPoints`<1000.

  [Note that the calculation in part d) may take 20 sec to run. If it takes substantially longer than that, it may be due to an error in your code.]
  [3 marks]

In[12]:=
```
NestWhile[updateIntegration, {0, 0, 0, 0}
 (*start with nothing and keep on updating*)
 , #[[1]] < 1000 & (*until number of points is 1000*)
]
```

Total Marks: 33

- **Total marks available: 33**

- **Solutions are due *by* 1200 noon on Thursday March 9th *here*: allow time for uploading on moodle.**
- A 10% mark deduction will be made (4 marks) if the template isn't used.
- Name your solution notebook file in the format **WK7_HMWK_<Initials>_<Family Name>.nb**, e.g. **WK7_HMWK_K_Hamilton.nb**
- Make a *backup copy* of your solutions.
- Delete all cell evaluation output by selecting **Cell → Delete All Output** from the drop-down menus at the top of the screen, then save and upload *that* file to Moodle.
- The first thing your marker will do when they receive your notebook is to evaluate all of it, to regenerate the output, by clicking **Evaluation → Evaluate Notebook** from the drop-down menus at the top of the screen. *It is your responsibility to check that carrying out this process, with a fresh kernel, will produce the output you intend it to, before you upload your work.*

K. Hamilton

15:26 21 Feb 2023