Computing for Mathematical Physics 2022/23

Homework8

Mark for homework8: /40

(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 three question headings to open and view them.

- 1. The shooting method. [20 marks]
  - It may help to quickly refresh your memory regarding the *shooting method*, revisiting the last question in this week's class exercises and/or the last section of this week's lecture notebook.
  - a) Code the following differential equation in *Mathematica*, assigning it the label `q1DifferentialEquation`:

    • eq. 1.1 : $\frac{d^2 y}{d t^2} + y^3 = 7\,\mathrm{Cos}[5\,t]$,

    Obtain a numerical solution of eq. 1.1 over the range -1 < t < 10, providing the boundary conditions `y[-1]=0` and `y[10]=0` to `NDSolve`, with `AccuracyGoal→8` and `PrecisionGoal→16`. Note: we are ***not*** employing the shooting method here in part a) yet, but rather supplying boundary conditions

y[-1]=0 and y[10]=0 *directly* to NDSolve instead. Plot the resulting solution for *y*(*t*) for -1 < *t* < 10.
[6 marks]

```
(*Create Equation*)
q1DifferentialEquation = y''[t] + y[t]^3 == 7 Cos[5 t];
```

In[50]:=
```
(*solve equation with boundary conditions*)
sol1 = NDSolve[{q1DifferentialEquation, y[-1] == 0, y[10] == 0},
   y[t], {t, -1, 10}, AccuracyGoal → 8, PrecisionGoal → 16];
```

- b) NDSolve, as used in part a), finds just *one of many* possible solutions to eq. 1.1. The function in the next input cell, q1Solver[slopeGuess_], is adapted from the answer to Q3 b) in week 8's in-class exercise on the *shooting method*: it returns a numerical solution for eq. 1.1 with boundary conditions y[-1]=0 and y′[-1]=slopeGuess. Analogous to Q3 c) in week 8's in-class exercise, define a function, yAttEquals10[slopeGuess_?NumericQ], returning the value of the y[t] solution output by q1Solver[slopeGuess][[1]] at *t* = 10. Use yAttEquals10 to plot the values of y[10] vs y′[-1] when solving eq. 1.1 with the boundary value y[-1]=0, and y′[-1] values between 0 and 1.
[4 marks]

In[6]:=
```
q1Solver[slopeGuess_] :=
 NDSolve[
  {
   (* Differential equation that we need to solve *)
   q1DifferentialEquation,
   (* Boundary condition y[-1]=0 from the question *)
   y[-1] == 0,
   (* Corresponding boundary condition for y'[t] at *)
   (* t = -1 which we set to the slopeGuess value   *)
   (* passed to q3Solver as an input.               *)
   y'[-1] == slopeGuess
  },
  (* Specify the function we want a solution for. *)
  y,
  (* Independent variable and the associated range *)
  (* over which we seek a numerical solution.      *)
  {t, -1, 10}
 ]
```

In[8]:=
```
yAttEquals10[slopeGuess_?NumericQ] :=
 (y[t] /. q1Solver[slopeGuess]〚1〛) /. t → 10
```

In[10]:=
```
Plot[(yAttEquals10[slope]), {slope, 0, 1}]
```

- c) Guided by the plot from part b), use FindRoot and yAttEquals10 to precisely determine the *two* values of y′[-1], between 0 and 1, corresponding to solutions of eq. 1.1 that satisfy y[-1]=0 and y[10]=0. Assign the solution corresponding

to the smallest `y'[-1]` value to the variable `slopeSolution1`. Assign the solution corresponding to the larger `y'[-1]` value to the variable `slopeSolution2`.
[4 marks]

```
In[1]:= (*Find solutions*)
slopeSolution1 = FindRoot[yAttEquals10[x], {x, 0}]
slopeSolution2 = FindRoot[yAttEquals10[x], {x, 0.65}]
```

- d) Solve eq. 1.1 using the boundary conditions `y[-1]=0` and `y'[-1]=slopeSolution1`, over the range -1 < *t* < 10, assigning the solution to the symbol `q1DSolution1`. Solve eq. 1.1 using the boundary conditions `y[-1]=0` and `y'[-1]=slopeSolution2`, over the range -1 < *t* < 10, assigning the solution to the symbol `q1DSolution2`. Plot these two solutions on the same graph, with due care given to its presentation.
[6 marks]

```
(*Solve for equations*)
q1DSolution1 = NDSolve[{q1DifferentialEquation,
   y[-1] == 0, y'[-1] == x /. slopeSolution1}, y[t], {t, -1, 10}]
q1DSolution2 = NDSolve[{q1DifferentialEquation,
   y[-1] == 0, y'[-1] == x /. slopeSolution2}, y[t], {t, -1, 10}]

(*Plot both functions*)
Plot[{y[t] /. q1DSolution1, y[t] /. q1DSolution2},
 {t, -1, 10}, Frame → True, AxesLabel → {"Time", "Y"},
 PlotLegends → {x /. slopeSolution1, x /. slopeSolution2},
 PlotLabel → "Differential Equation 1.1 solutions"]
```

- 2. Numerical solving equations of motion. [20 marks]

  - Two particles, each of unit mass in some system of units, move along the x axis and are joined by a spring of natural length *L* = 1 and spring constant *k* = 5. The particles are initially released *from rest* with one particle at x = 0.1 and the other at x = 0.9.

```
In[1]:= Clear["Global`*"]
```

- a) Code the masses of the two particles as `m1=1`, `m2=1`. Code the natural length of the spring as `L=1`, and the spring constant as `k=5`. Similarly, declare the maximum time period over which we will consider the evolution as `tMax=100`.

N.B. From part b) onward, the variables, `m1`, `m2`, `L`, `k`, and `tMax` should be used, whenever appropriate, to refer to those physical quantities in the code; instead of *explicitly* entering (hard-wiring) their corresponding numerical values in formulae.

[0 marks]

```
In[1]:= {m1, m2, L, k, tMax} = {1, 1, 1, 5, 100};
```

- b) Set up and solve, *numerically*, the equations of motion for the particles, from time `t=0` up to `t=tMax=100`. Plot the positions of the two particles as a function of time, on the same graph.
  [9 marks]

```
(*Set force equation*)
for[t_] := {A Cos[t[[1]] √(k / m)], t[[2]]}
(*Set position vectors*)
r1[t_] = {x1[t], y1[t]};
r2[t_] = {x2[t], y2[t]};
(*Set equations of motion*)
fspring1 = for[r1[t]];
fspring2 = for[r1[t]];
equat1 = m1 D[r1[t], {t, 2}] == fspring1 /. m → m1 /. A → (1 - 0.1)
equat2 = m2 D[r2[t], {t, 2}] == fspring2 /. m → m2 /. A → (1 - 0.9)

(*Set initial parameters*)
r10 = {0.1, 0};
r20 = {0.9, 0};
v10 = {0, 0};
v20 = {0, 0};
```

In[128]:=

```
(*Solve equations of motion for positions *)
q2Solution = NDSolve[
   Join[
    Map[Thread, {
      equat1,
      equat2,
      (D[r1[t], t] /. t → 0) == v10,
      (D[r2[t], t] /. t → 0) == v20,
      r1[0] == r10,
      r2[0] == r20}]],
   Join[r1[t], r2[t], {x1'[t], x2'[t]}],
   (*Create list of all parameters to solve for*)
   {t, 0, 100}];
```

In[112]:=
```
(*Plot results*)
ParametricPlot[
 Evaluate[
  {
    {x1[t], y1[t]},
    {x2[t], y2[t]}
   } /.q2Solution[[1]] (*Apply solution*)
 ],
 {t, 0, 100},
 PlotStyle → {Red, Green},
 AxesLabel → {"X position", "Y position"},
 PlotLegends → {0.1, 0.9},
 PlotRange → {{0, 1.5}, {-0.1, 0.1}}
]
```

- c) The kinetic energy of a single particle of mass m moving with velocity $\frac{dx}{dt}$ is $\frac{1}{2}\, m \left(\frac{dx}{dt}\right)^2$. The elastic energy in a spring of natural length L, and spring constant k, when its length is d, is $\frac{1}{2}\, k(L-d)^2$. Using this information, plot the total energy in the system as a function of time.
  [5 marks]

In[136]:=
```
(*Set equations of Energies*)
elasticPotential[t_] = 0.5 m (x1'[t])^2 /. m → 1 /. q2Solution;
kineticEnergy[t_] = 0.5 k (L - x1[t])^2 /. q2Solution;
elasticPotential2[t_] = 0.5 m (x2'[t])^2 /. m → 1 /. q2Solution;
kineticEnergy2[t_] = 0.5 k (L - x2[t])^2 /. q2Solution;
```

In[141]:=
```
Plot[{elasticPotential[t] + kineticEnergy[t] +
    elasticPotential2[t] + kineticEnergy2[t]},
 {t, 0, 100}, AxesLabel → {"Position", "Total Energy"}]
```

- d) Repeat b) and c) for the case in which the particles are released from rest at x=0.4 and x=0.6. Plot the positions of the two particles as a function of time, on the same graph. Also, separately, plot the total energy in the system as a function of time, choosing the range displayed so as to show any trends. Comment briefly on the latter plot in a text cell.
  [6 marks]

In[82]:=
```
equation1 = m1 D[r1[t], {t, 2}] == fspring1 /. m → m1 /. A → (1 - 0.4);
equation2 = m2 D[r2[t], {t, 2}] == fspring2 /. m → m2 /. A → (1 - 0.6);
```

In[143]:=
```
r10d = {0.4, 0};
r20d = {0.6, 0};
v10d  = {0, 0};
v20d  = {0, 0};
```

In[148]:=

```
q2dSolution = NDSolve[
   Join[
    Map[Thread, {
      equation1,
      equation2,
      (D[r1[t], t] /. t → 0) == v10d,
      (D[r2[t], t] /. t → 0) == v20d,
      r1[0] == r10d,
      r2[0] == r20d}]],
   Join[r1[t], r2[t], {x1'[t], x2'[t]}],
   (*Create list of all parameters to solve for*)
   {t, 0, 100}];
```

In[149]:=

```
ParametricPlot[
 Evaluate[
  {
    {x1[t], y1[t]},
    {x2[t], y2[t]}
   } /.q2dSolution(*Apply solution*)
 ],
 {t, 0, 100},
 PlotStyle → {Red, Green},
 AxesLabel → {"X position", "Y position"},
 PlotLegends → {0.4, 0.6},
 PlotRange → {{0, 1.5}, {-0.1, 0.1}}]
```

In[155]:=

```
elasticPotentialD[t_] = 0.5 m (x1'[t])^2 /. m → 1 /. q2dSolution;
kineticEnergyD[t_] = 0.5 k (L - x1[t])^2 /. q2dSolution;
elasticPotential2D[t_] = 0.5 m (x1'[t])^2 /. m → 1 /. q2dSolution;
kineticEnergy2D[t_] = 0.5 k (L - x1[t])^2 /. q2dSolution;
Plot[{elasticPotentialD[t] + kineticEnergyD[t] +
   elasticPotential2D[t] + kineticEnergy2D[t]},
 {t, 0, 100}, AxesLabel → {"Position", "Total Energy"}]
```

When the positions are changed, the total energy oscillates in the same fashion, but the peak is higher and the through reaches zero. The normal range shown shows all trends so there is no need to set a range.

- **Total marks available: 40**

- **Solutions are due *by* 1200 noon on Thursday March 16th *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 **WK8_HMWK_<Initials>_<Family Name>.nb**, e.g. **WK8_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 will produce the output you intend it to, before you upload your work.*

K. Hamilton

15:35 21 Feb 2023