Exercise: Synthesis exercise

In this exercise, we'll be playing a game of golf. The golf-ball is described as a point-mass in an X-Y plane with states for position $\vec{p}$ and velocity $\vec{v}$. Its dynamics are governed by:

$$\begin{cases} \dot{\vec{p}} = \vec{v} \\ \dot{\vec{v}} = \vec{g} - \frac{1}{m} c ||\vec{v}||_2 \vec{v} \end{cases} \tag{1}$$

Physical constants from the above ODE are given:

```
m = 0.04593  # ball mass [kg]
g = 9.81     # gravity [m/s^2]
c = 17.5e-5; # friction [kg/m]
```

1 Integrating the golf ball's state

Tasks:

1. Implement a 4-by-1 CasADi expression `rhs` for the ODE right-hand side:

   ```
   p = MX.sym('p',2);
   v = MX.sym('v',2);

   rhs = ... #  use norm_2(...)

   f = Function('rhs',[vertcat(p,v)],[rhs])
   ```

   Validate it by numerical evaluation:

   ```
   print(f([0.0,0.0,35.0,30.0])) # should be [35, 30, -6.14737, -15.0792]
   ```

2. Create a CasADi integrator as follows:

   ```
   ode = {'x':vertcat(p,v),'ode':rhs}

   options = dict();
   options["tf"] = 1

   intg = integrator('intg','cvodes',ode,options)
   ```

   Print out `intg`. Which inputs and outputs have a non-zero dimension? We are only interested in the `x0` input (initial state at $t_0 = 0\mathrm{s}$) and the `xf` output (integrated state at $t_f = 1\mathrm{s}$).

   Evaluate the `intg` CasADi Function numerically at `x0=[0.0,0.0,35.0,30.0]`, and verify that you obtain:

   ```
   xf =  [32.3549, 23.0661, 30.075, 16.6423]
   ```

2 Determining the reach of the golf ball

1. Create a CasADi Function `fly1sec` that maps an initial state to a final state 1 second later. Use a symbolic call to the `intg` Function.

   ```
   X0 = MX.sym('X0',4)
   fly1sec = Function('fly1sec',[X0],[...])
   ```

   Verify your implementation numerically:

   ```
   print(fly1sec([0.0,0.0,35.0,20.0])) # should be [32.6405, 13.9607, 30.5608, 8.26437
   ```

2. Instead of a fixed time interval, make the time interval T a second argument:

   ```
   fly = Function('fly',[X0,T],[...])
   ```

   Note that you will need to perform a time-transformation because the `tf` option cannot be symbolic: A dynamic system $\frac{dx}{dt} = f(x)$ to be integrated over $t = 0..T$ is equivalent to a system $\frac{dx}{d\tau} = Tf(x)$ to be integrated over a normalised time $\tau = 0..1$.

   Verify that you obtain for a flight time of 5 seconds:

   ```
   fly([0.0;0.0;35.0;30.0],5) % should be [130.338, 8.27205, 19.8961, -21.2868]
   ```

3. The full state-vector is a bit cumbersome to work with. Let's assume we always launch from $\vec{p} = \vec{0}$ and with a shooting speed v and shooting angle θ (degrees) that is defined as the positive angle wrt to the horizon.

   Create a CasADi Function `shoot` that provides us the full state using that parametrization for initial values:

   ```
   shoot = Function('shoot',[v,theta,T],[...])
   ```

   You may verify numerically with:

   ```
   shoot(50,30,5) % should be [155.243, -11.0833, 22.6012, -23.8282]
   ```

4. For $v = 50\text{m/s}$ and $\theta = 30\text{degree}$, find the flight time T that is needed for the ball to hit the ground. Use a CasADi rootfinder, and provide $T = 5\text{s}$ as initial guess.

   ```
   rf = rootfinder('rf','newton',{'x':...,'g':...}) # Solve g(x,p) = 0 for x
   ```

   Verify that you obtain $T^\star = 4.49773\text{s}$.

5. Flight time T is only an intermediate result to obtain total horizontal distance covered by the golf-ball at touchdown (=reach). Create a CasADi function `shoot_distance` that outputs this reach. This time, you'll need rootfinding in parametric form $g(x, p) = 0$.

   ```
   shoot_distance = Function('shoot_distance',[v,theta],[...])
   shoot_distance(50,30) # should be 143.533
   ```

3 Optimizing for shooting angle

1. For a launch speed of $v = 30\mathrm{m/s}$, find the launch angle that maximizes the distance covered. Use the `shoot_distance` CasADi Function and construct a CasADi nlpsol Function:

```
nlp = dict()
nlp["x"] = ...
nlp["f"] = ...

solver = nlpsol('solver','ipopt',nlp)

res = solver(x0=...)
```

   Verify that the result is `43.2223`. Why is it different from the $45\mathrm{degree}$ solution that you learn in high school?

2. No-one can shoot a golf-ball with infinitely accurate initial conditions.

   Let's say that the covariance in initial conditions for a novice player is $\mathrm{cov}\left(\begin{bmatrix} v \\ \theta \end{bmatrix}\right) = \begin{bmatrix} 1^2 & 0 \\ 0 & 1.2^2 \end{bmatrix}$.

   If the hole is 80 meters away from the launch position, find the speed-angle pair that maximizes the novice's chance for a hole in one. This is equivalent to minimizing the covariance of the shooting distance.

   Use an approximation for covariance of a function $F(x)$:

$$\mathrm{cov}\left(F(x)\right) \approx \frac{\partial F}{\partial x}\mathrm{cov}(x)\frac{\partial F}{\partial x}^T .$$

   For matrix multiplication, use `A@B` for Python 3.5 and above or `mtimes(A,B)` for lower versions.

   Your NLP solver will need a reasonable initial guess to work: $30\mathrm{m/s}$, and $45\mathrm{deg}$.

   Use the following boilerplate:

```
nlp = dict()
nlp["x"] = ...
nlp["g"] = ...
nlp["f"] = ...

solver = nlpsol('solver','ipopt',nlp)

res = solver(x0=...,lbg=...,ubg=...)
```

   Verify that the result is `[34.1584, 57.0871]`.