Project Manual Bachelor Year 1
Project 1.2
Crazy Putting!

Otti D'Huys, Philippe Dreesen, Nico Roos, Evgueni Smirnov

Period 1.4, 1.5, and 1.6
Academic year 2023-2024
Bachelor Data Science and Artificial Intelligence
Maastricht University

Courses:

Data Structures and Algorithms (KEN1420)
Principles of Data Science (KEN1435)
Calculus (KEN1440)
Linear Algebra (KEN1410)
Software Engineering (KEN1520)
Numerical Methods (KEN1540)
Computational and Cognitive Neuroscience (KEN1210)

1 Overview

The central topic of Project 1-2 is to create a simulator and artificial intelligence for putting on sloping greens with obstacles. In phase 1 you concentrate on building the underlying differential equation solvers, in later phases you use these to simulate a golf game.

2 Dynamical systems

2.1 Introduction

Dynamical systems, or systems that evolve in time are used to model phenomena ranging from neuroscience (how does a neuron react to a stimulus?) to ecology (how do populations of predators and preys influence each other?) or infectious disease spreading (how to model the evolution of a covid wave?). A dynamical system describes the evolution in time of a system with ordinary differential equations (ODEs): the time derivative describes the change in a variable, depending on current conditions.

Moreover, all laws of physics (dynamics, electrodynamics, quantum mechanics,...) are formulated as differential equations. Hence, modelling the time evolution of practically any real life system involves integrating differential equations.

However, while differential equations are a natural tool to build a model, they usually cannot be solved analytically. In phase 1, you build ODE solvers that approximate the solution numerically. You test them for reliability, and determine their optimal settings for the subsequent phases.

2.2 Building ODE solvers - phase 1

In the first phase, you concentrate on implementing ODE Solvers. Your code includes a basic GUI, an input module and, most importantly a module consisting of at least two different ODE solvers: the Euler differential equations solver described in Appendix A, and at least one higher order solver of your choice.

Tasks:

- Implement an ODE solver module, that can integrate systems of first order differential equations of an arbitrary dimension (up to 10).
- Implement an input module, allowing for the user to input parameters and initial conditions, and choose the step size, integration time and the solver that should be used.
- Create a GUI, that shows the evolution the variables in time and in phase space (plotting variables with respect to each other). The user should be able to specify which output they want to see.
- Compare the computation time and accuracy of your solvers for varying step size in a log-log plot. You should use a suitable ODE with an analytical solution for this comparison.

Minimal requirements for passing the phase:

- a fully correctly implemented Euler solver.
- a basic GUI that shows the numerical solution of an ODE.
- an implementation in which the ODE solver, the GUI, and the ODE system are independent.

- an experiment that tests the accuracy of your solver(s).

At the presentation, we will ask you to run your code with a new system of differential equations (make sure that this can easily be implemented, through the GUI as user input, or in the directly in the code!), and we will verify your output.

2.3 Suggested test systems

We suggest that you use one of the following well known systems for testing your solvers. (Note that these are not suitable to test the accuracy, as they do not always have an analytical solution).

1. The **Lotka-Volterra equations** for population dynamics [1, 2],

$$\begin{aligned} \dot{x} &= \alpha x - \beta xy \\ \dot{y} &= \delta xy - \gamma y \end{aligned} \tag{1}$$

   Here $x(t)$ denotes a population of prey, and $y(t)$ a population of predators, and the dot denotes the time derivative, $\dot{x} = \frac{dx}{dt}$. The number of prey x grows proportionally to their own number (they multiply), but the prey get killed by the predators at a rate βxy (the more predators and the more prey, the more killings). The population of predators $y(t)$ declines in the absence of prey (they starve), this is the term $-\gamma y$), and they grow proportional to the amount of food (the term δxy) . This system represents one of the earliest models in mathematical ecology, for an introduction we refer to the Scholarpedia page (`http://www.scholarpedia.org/article/Predator-prey_model`).

   The parameters $\alpha, \beta, \gamma, \delta$ and the variables $x(t)$ and $y(t)$ are all positive.

2. The **FitzHugh-Nagumo model** for a spiking neuron [3]. This is a simplified model that shows excitable dynamics (i.e. spiking)

$$\begin{aligned} \dot{V} &= V - \frac{V^3}{3} + I_{ext} \\ \dot{W} &= \epsilon(V + a - bW) \end{aligned} \tag{2}$$

   Here, $V(t)$ represents the membrane voltage of the neuron, the slower, linear recovery variable $w(t)$ represents sodium channel reactivation and potassium channel deactivation and I_{ext} is proportional to an additional current due to a stimulus. This model is not biologically accurate, but it allows an easy mathematical explanation of the spike-generating mechanism. You can find more information on `http://www.scholarpedia.org/article/FitzHugh-Nagumo_model`.

   The parameters a, b and ϵ are positive, and ϵ is small. Typical values: $\epsilon \approx 0.05$, $a \approx b \approx 1$

3. The **SIR-model** to predict the spread of infectious diseases [4]. The population is divided in three compartments: $S(t)$ is the susceptible fraction of the population, $I(t)$ is the fraction of the infected individuals, and $R(t)$ is the fraction that has recovered and have acquired immunity.

$$\begin{aligned} \dot{S} &= -kSI + \mu(1 - S) \\ \dot{I} &= kSI - (\gamma + \mu)I \\ \dot{R} &= \gamma I - \mu R \end{aligned} \tag{3}$$

   In this model, susceptible people get infected, and infected people recover, $S \to I \to R$. The susceptible population becomes infected at a rate kI, the proportionality

constant k depends on the average number of contacts, and how contagious the disease is. The recovery rate γ, which models the transition from infected individuals I to recovered and immune individuals R, is inversely proportional to the duration of the disease. People die at a rate μ (μ is inversely proportional to the lifetime of an individual), and babies are born susceptible at the same rate.

Compartment models like the SIR model have been used to model and predict infection outbreaks, as flu, measles and common cold since the early 20th century. Currently several Covid prediction models are based on (extended) SIR models.

The parameters k, γ and μ are all positive. Choose values $k, \gamma \in [1, 5]$, $\mu \approx 0.001$.

3 Golf simulation

3.1 Putting

In phases 2 and 3, you create a putting simulation program using the ODE solvers that you have built in phase 1. Putting is an important part of one of the world's most popular and exciting sports, namely golf [5]. In putting, players aim to strike a ball across a grassy green field to have it fall into a hole. This is made difficult by the fact that the green is not necessarily flat, requiring the player to accurately judge how the slope will affect the motion of the ball. In some cases, the green may also contain obstacles such as sand pits, trees or even water.

Your program should capture all the excitement of the real-life putting experience from the comfort of your room. The program should approximate the physics of real-life putting, and allow for players to play against each other or an artificial intelligence player (bot). The program should also allow users to create their own putting greens, complete with gentle rolling hills, obstacles and lakes.

3.1.1 Physics

The motion of the golf ball is governed by Newton's second law (a second order differential equation). Each time it is hit, the ball is set in motion in a given direction. The speed is determined by how hard the ball is hit, up to some given maximum speed $v_{\max}$.

To simplify the physics we assume that the ball is always sliding on top of the terrain, and is always hit along the surface (not upwards). You do not have to simulate the rolling motion of a ball, or consider flying motion. You can also use the approximation that the slopes are not very steep, and only change gently.

The motion of the ball across the course is governed by three forces:

1. The force of gravity. The gravitational force is constant, and is directed downwards.
2. The normal force[1]. As the name indicates, this force is directed normal to the surface. Thus, its magnitude depends on the slopes of the terrain. Due to sum of the gravitational and normal force, the ball accelerates when it moves downhill, and decelerates if it slides uphill.
3. The force of friction. When the ball is moving, we speak of kinetic friction. The kinetic friction is always directed opposite to the direction of motion. The magnitude of the kinetic friction is proportional to the normal force, with a proportionality constant μ_K.
   When the ball comes to rest (on a slope), we speak of static friction. The static friction force might prevent the ball from sliding down - this happens when the friction compensates for the downhill force. The maximal static friction force is also proportional to the normal force, the proportionality constant μ_S (the static friction coefficient) is larger than the kinetic friction coefficient $\mu_K < \mu_S$.

The system of differential equations the models the motion of the ball is given in the appendices.

3.1.2 Terrain

The terrain is described by a function h giving the height profile $z = h(x, y)$ at every point, and by the coefficients of friction μ_S and μ_K. Further, the terrain can have bodies

[1] The normal force is a force exerted by the earth on the golf ball, as a reaction force to the force that the golf ball exerts on the earth due to its weight

of water, which are found where the height z is negative. You may optionally allow for additional obstacles, such as sand pits (which have higher friction coefficients), and trees (hitting a tree incurs the same penalty as falling into the water. You do not need to consider any bouncing back).

3.1.3 The Game

The ball has to make its way across the green from its starting point to the target, while avoiding any pools of water. For the purposes of this game, instead of a hole, the player aims to strike the ball so that it stops within a given radius r of a marked target.

The player aims to guide the ball into the target in as few shots as possible. If the ball falls into the water, the player incurs a one-stroke penalty, and replays the shot from the initial location.See [1, Rule 26] for the full regulations.

3.2 The Software

The programming language for this project is **Java**. You are not allowed to use an existing physics engine or mathematical solvers, but must implement these yourself. However, you are allowed to use an existing graphics library.

The final software should meet the following requirements:

A. Object-Oriented Design: The software must utilize an object-oriented design approach. This includes the use of classes and objects, inheritance, polymorphism, encapsulation, and abstraction. Design your software with clear class diagrams and interfaces, ensuring that each class has a single responsibility and promotes code reuse.

B. Procedural Programming Principles: Where appropriate, procedural programming techniques should be used for defining functions, managing control flow, and implementing algorithms that do not naturally fit into an object-oriented paradigm. This includes the use of loops, conditionals, and function calls to manage the game's logic and interactions.

C. Data Structures Utilization: Implement and leverage appropriate data structures learned throughout the courses, such as arrays, lists, stacks, queues, trees, and graphs. These could be used for managing the game elements (e.g., obstacles, terrain), implementing game mechanics (e.g., pathfinding for the AI), and optimizing performance.

D. Software Engineering Practices: The project should reflect best practices in software engineering. This includes but is not limited to:

- Version Control: Use Git for source code management, with proper commit messages and branching strategies.
- Modular Design: Organize the code into modules or packages based on functionality.
- Documentation: Document the code extensively, including class and method descriptions, to ensure maintainability and readability.
- Testing: Implement unit tests for critical components of the game to validate functionality and identify bugs early in the development process.

  - Design Patterns: Apply relevant design patterns to solve common software design problems.

E. Graphical User Interface (GUI): Create a user-friendly GUI that allows players to interact with the game, including starting the game, inputting commands or actions, and visualizing the game state. The GUI should be intuitive and provide feedback to the player on the game's progress and outcomes.

F. Performance Considerations: The software should be optimized for performance, ensuring that the game runs smoothly without unnecessary delays or resource consumption. This includes optimizing data structures and algorithms for efficiency.

G. Scalability: Design the software with scalability in mind, allowing for easy addition of new features, such as new terrain types, obstacles, or game modes, without significant restructuring of the code base.

3.3 Implementation of a putting game - phase 2

In the second phase you apply your ODE solvers created in phase 1 to implement a physics engine for the crazy putting game. Secondly you implement an artificial intelligence player capable of analyzing the course and computing how to strike the ball.
Tasks:

- Implement a physics engine for the crazy putting game. The engine must be able to handle sloping courses following the equations of motion given in Appendix B.Use a 4-dimensional state vector
$$\mathbf{x}(t) = \begin{pmatrix} x(t) \\ y(t) \\ v_x(t) \\ v_y(t) \end{pmatrix}.$$
- Implement a course input module, allowing for input from a file, or from a user. The input should specify
  - the height profile $h(x, y)$ by a specific function (e.g. $h(x, y) = \sin\left(\frac{x-y}{7}\right) + 0.5$). Negative heights denote lakes, so you do not need to implement those separately.
  - the friction coefficients of the green.
  - the position (x_t, y_t) and radius r of the target.
  - the start position (x_0, y_0).
  - (optional in phase 1) the location and radius of any trees.
  - (optional) the location and the friction coefficients of any sand pits. (Example: the inequalities $3 < x < 5, 0 < y < 3$ represent a rectangular sand pit of 2m by 3m.)
- Implement a simulator where the initial velocity $\mathbf{v}_0 = (v_{0x}, v_{0y})$ of each shot can be specified by the user.
- Create a basic GUI, that shows your green, and the position of the ball. After each shot, the coordinates of the position (x_f, y_f) where the ball comes to rest,and the number of shots so far, should be shown on your screen in your GUI.
- Create additional higher order ODE solvers, such as Runge-Kutta 4 or Verlet. Compare their accuracy to the solvers that you created in phase 1. If you did not have a correct higher order solver in phase 1, you do should create one in phase 2.

- Add a basic (rule-based) bot. The bot may have full knowledge of the geometry and slope of the terrain.
- Add an artificial intelligence player capable of playing on the terrain you create, and compare this bot to the basic bot. This bot should consistently manage to score a hole-in-one, where possible, but does not need (yet) to handle complex, maze-like courses. Hint: Try Hill-Climbing or Newton-Raphson and only then more advanced approaches. Genetic algorithms may not be the best option to try.
- Implement a collision-detector, for determining whether the ball has fallen into a body of water. Show that your AI player is able to avoid obstacles by putting at least one lake in the course, at a relevant location. Sand pits, and multiple obstacles are optional, but encouraged. It is not necessary that your GUI allows to create and place these obstacles on the course, they can be imported from a file.
- You submit a first draft of the project report by the end of phase 2. You will receive feedback on your draft during the pre-examination of phase 3.

In order to pass the second phase, you need to have at least one correctly working higher-order solver. We require a rule-based bot and an intelligent bot, that scores a hole-in-one when possible.

To test your bot, try to score a hole-in-one on the following test course, using the given parameters:
$h(x,y) = 0.4(0.9 - e^{-\frac{x^2+y^2}{8}}), \mu_K = 0.08, \mu_S = 0.2, (x_0, y_0) = (-3, 0), (x_t, y_t) = (4, 1), r = 0.15$
The parameters and parameter ranges relevant to your physics engine are given in Table 1.

Table 1: Parameters, parameter ranges and restrictions.

Quantity	Symbol	value/range	restrictions
Course profile	$h(x,y)$	-10m - 10m	$\left\lvert\frac{\partial h}{\partial x}\right\rvert, \left\lvert\frac{\partial h}{\partial y}\right\rvert \leq 0.15$, $\left\lvert\frac{\partial^2 h}{\partial x^2}\right\rvert, \left\lvert\frac{\partial^2 h}{\partial x \partial y}\right\rvert, \left\lvert\frac{\partial^2 h}{\partial y^2}\right\rvert \leq 0.1$/m
Mass of the golf ball	m	0.0459 kg	
Gravitational constant	g	9.81 m/s^2	
Kinetic friction (grass)	μ_K	0.05-0.1	
Static friction (grass)	μ_S	0.1-0.2	$\mu_S > \mu_K > 0$
Kinetic friction (sand)	$\mu_{K,s}$		$0 < \mu_K < \mu_{K,s} < 1$
Static friction (sand)	$\mu_{S,s}$		$0 < \mu_{K,s}, \mu_S < \mu_{S,s} < 1$
Maximum speed	$v_{\max}$	5 m/s	
Target radius	r	0.05m - 0.15m	

3.4 Advanced Topics - Phase 3

In the third phase you focus on improvements to the AI and look at some extensions for the game. If your solvers or intelligent bot from phase 2 do not yet work correctly, you should first fix these.

Choose one or more of the following tasks:

(a) Extend your AI bot player to handle complex maze-like courses. Design and test your AI on courses which allow you to identify the limits of your bots capabilities. You should not be able to reach your goal in less than three shots

(b) Humans are not perfect, and neither should your machine be. Introduce a small random error in the initial position and velocity of the ball and investigate how this affects performance of your AI. Vary the noise and investigate how your AI can find strategies to deal with this noise. Can you design a course for which it is better for the bot to play with a different strategy to the noise-free case?

(c) The equations of motion are only a first order approximation. Implement the more complete physical description given in Appendix C. Compare both models: when and how do they differ? Here, you are allowed a course profile with steeper hills.

As minimal requirements for the product in the third phase, you should have completed the tasks of the second phase correctly (i.e. you have a correctly implemented higher-order solver, and an intelligent bot that can handle obstacles), and you have meaningful results for at least one of options of phase 3.

References

[1] A.J. Lotka, Elements of physical biology. Williams and Wilkins, Baltimore, 1925

[2] V. Volterra (1926), Variazioni e fluttuazioni del numero d'individui in specie animali conviventi, *Mem. R. Accad. Naz. dei Lincei 2* 31, 113

[3] FitzHugh R. (1955) Mathematical models of threshold phenomena in the nerve membrane, *Bull. Math. Biophysic*, 17:257–278

[4] Kermack WO, McKendrick AG (1927), A Contribution to the Mathematical Theory of Epidemics, *Proceedings of the Royal Society of London. Series A*, 115 (772): 700–721.

[5] United States Golf Association (USGA) Rules and Decisions, `http://www.usga.org/content/usga/home-page/rules/rules-and-decisions.html`

[6] J. Tierney (2019). Better Science Through Mini-Golf? *New York Times* `http://tierneylab.blogs.nytimes.com/2009/06/15/better-science-through-mini-golf/`

[7] R.C. Hibbeler (2007) Engineering Mechanics: Statics & Dynamics, Pearson, 2007.

[8] J. Hierrezuelo and C. Carnero (1995) Sliding and rolling: The physics of a rolling ball, *Physics Educ. 30*, 177–182.

[9] A.R. Penner (2002). The physics of putting, *Canadian. J. Physics 80.*

[10] F. González-Cataldo, G. Gutiérrez and J.M. Yáñez (2017). Sliding down an arbitrary curve in the presence of friction, *Am. J. Physics 85*(2).

[11] R.A. Adams and C. Essex (2018), Calculus: A Complete Course, 9th edtion, Addison-Wesley

[12] Weir and Hass (2014). Thomas' Calculus. Early Transcendentals, 13th edition, Pearson

[13] J.D. Faires and R. Burden (2013), Numerical Methods, Brookes-Cole.

A Euler's method

In the Numerical Methods course, you will learn techniques for solving differential equations.

You need to solve a system of differential equations $(\dot{\mathbf{x}}) = \mathbf{f}(\mathbf{x})$, with $\mathbf{x}$ being your state vector

$$\mathbf{x}(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \end{pmatrix}$$

and $\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \end{pmatrix}$ describes the time evolution.
Then the updated state vector after a single time step of length h is

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{f}(\mathbf{x}(t)) = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \end{pmatrix} + h \begin{pmatrix} f_1(\mathbf{x}(t)) \\ f_2(\mathbf{x}(t)) \\ \vdots \end{pmatrix}$$

When you deal with second order equations (in phases 2 and 3), you rewrite the system as a set of first order equations: the second order equations

$$\begin{aligned} \ddot{x} &= f_x(x, y, \dot{x}, \dot{y}) \\ \ddot{y} &= f_y(x, y, \dot{x}, \dot{y}) \,, \end{aligned}$$

can be rewritten as a system of four first order equations,

$$\begin{aligned} \dot{x} &= v_x \\ \dot{y} &= v_y \\ \dot{v_x} &= f_x(x, y, v_x, v_y) \\ \dot{v_y} &= f_y(x, y, v_x, v_y) \,. \end{aligned}$$

Hence, we have an ODE system of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, with a state vector $\mathbf{x} = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix}$ and a system function $\mathbf{f}(\mathbf{x}) = \begin{pmatrix} v_x \\ v_y \\ f_x(x, y, v_x, v_y) \\ f_y(x, y, v_x, v_y) \end{pmatrix}$.

B Physics

The ball is moving through a 3-dimensional space. We describe its coordinates by a position (x, y, z) in a Cartesian coordinate system. We can describe the surface by a course profile specifying the height $z = h(x, y)$ as function of the x and y coordinate. Assuming that the ball always rests on the surface ($\dot{z} = \frac{\partial h}{\partial x}\dot{x} + \frac{\partial h}{\partial y}\dot{y}$), it is sufficient to only consider the x and y coordinate of the ball position (x, y). We need the velocity and acceleration in the x and y direction: $\mathbf{v} = (v_x, v_y) = (\dot{x}, \dot{y})$ and $\mathbf{a} = \dot{\mathbf{v}} = (\ddot{x}, \ddot{y})$.

When the ball is moving, the motion of the ball can be approximated by the following equations,

$$\ddot{x} = -g\frac{\partial h}{\partial x} - \mu_K g \frac{v_x}{\sqrt{v_x^2 + v_y^2}} \tag{4}$$

$$\ddot{y} = -g\frac{\partial h}{\partial y} - \mu_K g \frac{v_y}{\sqrt{v_x^2 + v_y^2}} \tag{5}$$

Here, the first term models the downhill force due to the combination of the normal force and gravity. The second term models the kinetic friction, which is always opposite to the motion.

When the ball comes to rest, you need to evaluate whether it will stay in rest, or slide down a hill. When $\frac{\partial h}{\partial x} = \frac{\partial h}{\partial y} = 0$, the ball is not on a slope, and it stays in rest. If $\frac{\partial h}{\partial x} \neq 0$, or $\frac{\partial h}{\partial y} \neq 0$, the ball stays in rest if the maximal static friction overcomes the downhill force, $\mu_S > \sqrt{\left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2}$.

Hint: if the ball does not stay in rest, the zero denominator of the friction force in Eqs. (1)-(2) can cause problems. When this occurs, you can adapt your solver in the following way: at zero speed, the ball starts moving in the direction of the acceleration (i.e. the slope), and thus the friction has a direction opposite to the horizontal components of the normal force. The x and y components for the friction (second term in Eqs. (1)-(2)) become $\frac{f_x}{m} = \mu_K g \frac{\frac{\partial h}{\partial x}}{\sqrt{\left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2}}$, $\frac{f_y}{m} = \mu_K g \frac{\frac{\partial h}{\partial y}}{\sqrt{\left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2}}$

C Derivation of the equations of motion

Three forces impact the golf ball. The gravitational force $\mathbf{G}$ is in the negative z-direction, and has a constant magnitude,

$$\mathbf{G} = -mg \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

The normal force $\mathbf{N}$ has a variable magnitude N, and is directed along the normal to the surface. Denoting unit vector normal to the surface as $\mathbf{e}_n$, we can write

$$\mathbf{N} = N\mathbf{e}_n, \text{ with } \mathbf{e}_n = \frac{1}{\sqrt{1 + \left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2}} \begin{pmatrix} -\frac{\partial h}{\partial x} \\ -\frac{\partial h}{\partial y} \\ 1 \end{pmatrix}.$$

The kinetic friction force is proportional in magnitude to the normal force, and is always directed along the surface and against the motion,

$$\mathbf{f} = -\frac{\mu_K N}{\sqrt{v_x^2 + v_y^2 + \left(\frac{\partial h}{\partial x}v_x + \frac{\partial h}{\partial y}v_y\right)^2}} \begin{pmatrix} v_x \\ v_y \\ \frac{\partial h}{\partial x}v_x + \frac{\partial h}{\partial y}v_y \end{pmatrix}.$$

Assuming no acceleration normal to the surface (which is justified if $\frac{\partial^2 h}{\partial x^2}$, $\frac{\partial^2 h}{\partial x \partial y}$, $\frac{\partial^2 h}{\partial y^2}$ are small), the normal component of the gravitational force equals the normal force:

$$N = -\mathbf{G} \cdot \mathbf{e}_n = \frac{mg}{\sqrt{1 + \left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2}}.$$

This results in a normal force

$$\mathbf{N} = \frac{mg}{1+\left(\frac{\partial h}{\partial x}\right)^2+\left(\frac{\partial h}{\partial y}\right)^2}\begin{pmatrix} -\frac{\partial h}{\partial x} \\ -\frac{\partial h}{\partial y} \\ 1 \end{pmatrix}.$$

We find for the friction

$$\mathbf{f} = -\frac{\mu_K mg}{\sqrt{1+\left(\frac{\partial h}{\partial x}\right)^2+\left(\frac{\partial h}{\partial y}\right)^2}\sqrt{v_x^2+v_y^2+\left(\frac{\partial h}{\partial x}v_x+\frac{\partial h}{\partial y}v_y\right)^2}}\begin{pmatrix} v_x \\ v_y \\ \frac{\partial h}{\partial x}v_x+\frac{\partial h}{\partial y}v_y \end{pmatrix}.$$

To find the equations of motion in x and y directions, we simply add the x and y components of normal force and the friction,

$$m\ddot{x} = -\frac{mg\frac{\partial h}{\partial x}}{1+\left(\frac{\partial h}{\partial x}\right)^2+\left(\frac{\partial h}{\partial y}\right)^2} - \frac{\mu_K mg}{\sqrt{1+\left(\frac{\partial h}{\partial x}\right)^2+\left(\frac{\partial h}{\partial y}\right)^2}}\frac{v_x}{\sqrt{v_x^2+v_y^2+\left(\frac{\partial h}{\partial x}v_x+\frac{\partial h}{\partial y}v_y\right)^2}} \quad (6)$$

$$m\ddot{y} = -\frac{mg\frac{\partial h}{\partial y}}{1+\left(\frac{\partial h}{\partial x}\right)^2+\left(\frac{\partial h}{\partial y}\right)^2} - \frac{\mu_K mg}{\sqrt{1+\left(\frac{\partial h}{\partial x}\right)^2+\left(\frac{\partial h}{\partial y}\right)^2}}\frac{v_y}{\sqrt{v_x^2+v_y^2+\left(\frac{\partial h}{\partial x}v_x+\frac{\partial h}{\partial y}v_y\right)^2}} \quad (7)$$

If we divide by the mass m of the golf ball, and use the assumption of not too steep hills $\left(\frac{\partial h}{\partial x}\right)^2, \left(\frac{\partial h}{\partial y}\right)^2 \ll 1$ and discard these terms in the denominators, we find Eqs. (1)-(2).

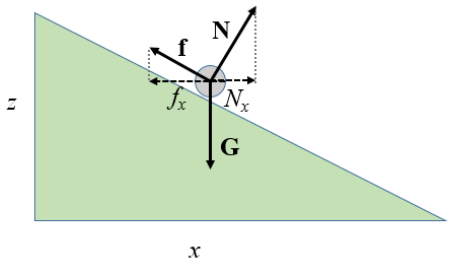

Figure 1: Free body diagram of the three forces governing the motion of a golf ball sliding down a slope.