# STEM Games 2018 - Mathematics Arena

Josip Žubrinić, Andro Merćep, Maja Jurišić Bellotti, Petra Gospodnetić, Mateja Đumić

In a small galaxy far, far away, with a small meaningless sun, located on a third rock from the sun, in a small country with the sea, in a small place on a tiny peninsula, a group of STEManics gathered having one clear aim – to save their loved ones and themselves in three days. If they do not find a way, STEMartians will destroy the meaningless Earth, Poreč and STEM-lovers. The only way to avoid the catastrophe and subside STEMartians' rage is to bribe their government. A team of skilled mathematicians has three days to find a way to use an aircraft in order to deliver a special package to the STEMartians base. The package contains perfectly salted deep-fried peanuts STEMartians would do anything for.

STEManiacs' job will not be easy. Only the most successful of them will be able to do differential equations, temporal discretization, strange weather conditions and other obstacles waiting for them in the galaxy far, far away. Leave all your prejudices because the Earth physical laws cannot be enforced in the galaxy, SI system does not exist and units of measurement do not have measurements. Hence, follow the provided instructions, beware of mice and STEMartians' poetry, bring a towel and turn your superbrain on because not everyone has a chance of saving the Earth. Your task is to model an aircraft by using differential equations (luckily, they do exist on STEMars) and implement a numerical method for the equations. Using the implemented solution, you need to plan an engine turning on in order for the aircraft to successfully complete the mission.

# Day 1 - Mathematical aircraft

## Introduction

The situation on STEMars is boiling! They really need your help! Our best scientists and spies gave their best or died trying to provide you with relevant information. You need to send them the aircraft and save them. It needs to defy wind, dust storms and acid rain in the STEMars atmosphere. The only way to achieve that is to design a correct mathematical flight model by using differential equations. Needless to say, prior to launching, make sure that the aircraft will actually reach its destination. Conduct a series of tests to check that. Likewise every important mission, this one needs to have a memorable name that will be written in the history of the universe so come up with a creative name for your aircraft.

## Aircraft

Model and calculate the aircraft motion through difficult conditions from the starting until destination point. The aircraft flies in a two-dimensional rectangular coordinated system. It is controlled by an engine which has an option of taking turns in different fixed positions. The aircraft motion is a vector function $x : [0, T] \to \mathbb{R}^2$, which is in line with the second Newton law. The speed is defined by vector $y(t) = \frac{d}{dt}x(t) = \dot{x}(t)$.

The aircraft has a limited volume gas tank filled prior to taking off and spent in an operating mode. The fuel mass in the tank in moment $t$ is defined by function $m_F(t)$. When the fuel is entirely used, the aircraft cannot be flown. It is coated with a thick layer of plating which protects it from the atmospheric weathering conditions. Aggressive weathering conditions damage the plating making the aircraft's mass volatile. When the plating is entirely used/damaged, the aircraft is destroyed. The plating mass in moment t is defined by function $m_S(t)$. The rest of the aircraft's mass is non-volatile. It is defined as $m_A$ so the aircraft's mass in moment $t$ equals:

$$m(t) = m_A + m_F(t) + m_S(t).$$

## Weathering conditions

Sandy clouds causing strong friction, acid rains damaging the plating and strong wind affecting the aircraft's flying direction can occur in the atmosphere. The wind speed is defined by function$w : \mathbb{R}^2 \to \mathbb{R}^2$, while acid rain and sandy storms are defined by densities $\rho_{\text{rain}} : \mathbb{R}^2 \to \mathbb{R}$ and $\rho_{\text{sand}} : \mathbb{R}^2 \to \mathbb{R}$. The ground is defined by function Ground which describes a relief's profile. The flight finishes in the moment of the aircraft touching the ground.

## Plan of starting the engine

Operating force$F_{\text{boost}}$ je is defined by a direction (an angle of engine turning) and amount. Engines are started according to the table *boost plan* which defines the starting moment $t_1, \ldots, t_n$ , corresponding intensity of an engine that is turning on $B_1, \ldots, B_n$ and its turning. The aircraft has only one engine. The plan of staring the engine is provided in table *boost plan* which lists moments $t_1, \ldots, t_n$ when engines are turned on and amounts of forces (intensities) $B_1, \ldots, B_n$ and turning of engines $k_1, \ldots, k_n$ which are turning on. Every engine turning on lasts for exactly 1 second or until the next input. An example of turning an engine on is provided in table 1.

Parameter *Engine shift (k)* in table 1 is an angle of engine's turning which is defined by formula:

$$\alpha_{\text{engine}} = \frac{k \cdot \pi}{4},$$

where the angle of the engine is defined with respect to the horizontal axis (abscise).

Table 1: Example of inputing in table boost plan.

| Indeks unosa | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Time ($t_i$) | 1 | 3 | 3.5 | 5 |
| Intensity ($B_i$) | 50000 | 100000 | 60000 | 80000 |
| Engine shift ($k_i$) | 6 | 2 | 6 | 4 |

There might be some overlapping which are to be solved in a way that a more recent input (the one with a higher index) has a higher priority. Consequently, all turning on the engines will last for 1 second; in time 3.5, index 3 will be active and not index 2.

### 1.1 Modelling

The total force on the aircraft is the sum of all forces affecting the aircraft. The forces of the environment affecting the aircraft are the following:

- Gravity force of planet $F_g$ whose gravity constant is defined by $g$ and is proportional to the mass of the aircraft $m(t)$. This force works in a vertical direction (ordinate axis).
- Resistive force through sandy clouds Fsand whose direction is the opposite of the velocity of the aircraft and proportional to the density of a sandy cloud in position $x(t)$ and aircraft's speed $y(t)$. The proportionality constant is $\gamma > 0$.
- Force of wind Fwind affecting the aircraft is proportional to the difference of wind speed in position $x(t)$ and current aircraft's speed $x(t)$. The proportionality constant is $\delta > 0$.

The engine force in moment $t$ is calculated using table *boost plan*. For moment $t$, an active input in the table has to be checked. In the case there are no active inputs in moment $t$, the engine force equals 0. As previously mentioned, in the case there are multiple inputs for moment $t$, the most recent input is the active one.

$$F_{\text{engine}} = B(t) \cdot e(t), \tag{1}$$

where functions $B(t)$ (amount of the force in moment $t$) and $e(t)$ (vector force in moment $t$) are read from the table.

When turning on the engine, fuel is used at the speed which is proportional to $B(t)$ and proportionality constant $\kappa > 0$. For every turning on the engine, fuel is used for exactly 1 second as defined in the boost plan unless there is an overlap, when this period is shorter.

When flying through acid rain, the plating is melted. The speed of melting is proportional to the density of acid rain clouds rain in position $x(t)$ with proportionality constant $> 0$. When the entiner plating is damaged, the aircraft is destroyed and stops flying.

The equation includes the starting requirements of launching on the speed and starting position.

## Exercise 1: Equation system

**Write a system of second order differential equations for motion $x(t)$ in the closed form $F(t, x(t), \dot{x}(t), \ddot{x}(t)) = 0$, $t > 0$,which is derived from the second Newton law.**

# Exercise 2: Exact solutions

The first step in testing of a model requires calculating exact solutions of a system in simple cases. Solve the given system in the following situations: mass of the aircraft is $m_A = 1000$, starting fuel mass $m_F(0) = 1000$, starting plating mass $m_S(0) = 1000$. A gravitation constant is $g = 5$. In moment $t = 0$, the aircraft is in position $x(0) = 0$ and is launched with the starting speed of 30 under angle $\pi/3$.

## 1.2 Situation 1

The aircraft flies solely in the gravity field having density $\rho_{\text{sand}} = \rho_{\text{rain}} = 0$and constant $\delta = 0$. Calculate its motion and determine the moment of landing.

## 1.3 Situation 2

The aircraft flies in the gravity field through sandy clouds with constant density $\rho_{\text{sand}}(x) = 50$,and constant friction $\gamma = 10$. Calculate its motion. ($\delta = 0$, $\rho_{\text{rain}} = 0$)

## 1.4 Situation 3

The aircraft flies in the gravity field through acid rain clouds with constant density $\rho_{\text{rain}}(x) = 500$, and constant plating consumption $\beta = 0.1$ ($\delta = 0$, $\rho_{\text{sand}} = 0$). Calculate its motion.

## 1.5 Situation 4

The aircraft flies in the gravity field pursuant to the *boost plan* provided in the following table:

| Input index | 1 |
|---|---|
| Time ($t_i$) | 5 |
| Intensity ($B_i$) | 30000 |
| Engine shift ($k_i$) | 2 |

($\delta = 0$, $\rho_{\text{sand}} = 0$, $\rho_{\text{rain}} = 0$, $\kappa = 0.01$). Calculate its motion up to moment $t = 6$.

## 1.6 Situation 5

Calculate the trajectory of the aircraft which swings in the wind field having constant speed $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$, $\delta > 0$ thus disregarding other forces. Explain what happens to the aircraft's speed when $t \to \infty$.

## 1.7 Situation 6

In moment $t = 0$ , the aircraft is launched $x_2(0) = 0$ vertically up having the speed of $y_2(0) = 15$through a sand cloud of density $\rho_{\text{sand}}(x_2) = 300x_2$. All forces except $F_{\text{sand}}$ are to be disregarded. Calculate its trajectory and explain what happens with the aircraft when $t \to \infty$.

## Discretization of differential equations

The previous system of differential equations:

$$F(t, x(t), \dot{x}(t), \ddot{x}(t)) = 0, \quad t > 0 \tag{2}$$

is solved numerically.

The approach to solving differential equations is deducing differential equations to a series of algebraic equations (difference equation). The first step is to carry out domain discretization during which we introduce a discretization step $h > 0$ and define a series of time moments $t_0 = 0, t_1 = h, t_2 = 2h, \ldots, t_i = ih$. Discretization is used in order to replace the entire functions of solutions with just a series $x_0, x_1, x_2, \ldots$, which represents the approximation of a function value $x(t)$ in discretization periods. To paraphrase, we are looking for series $x_k$ such as:

$$x_k \approx x(t_k).$$

Since this is the second order equation (2), we have to reduce it to a first order equation for the sake of more simple exercise solving, i.e. for using one-step methods. By introducing a new variable:

$$y(t) := \dot{x}(t)$$

we reduce the equation (2) to the system:

$$\begin{aligned} F(t, x(t), y(t), \dot{y}(t)) = 0, \quad t > 0 \\ \dot{x}(t) - y(t) = 0. \end{aligned} \tag{3}$$

The system (3) can be written with the formula:

$$G(t, z(t), \dot{z}(t)) = 0, \quad t > 0, \quad z(t) = (x(t), y(t)). \tag{4}$$

The next step is to introduce discretization methods for derivations and integrals. As a discretization scheme for derivations, we propose finite backward difference approximation. For integral approximation, we propose trapezoidal rule approximation. Using the aforementioned approximations, a system of differential equations (4) can be written as a series of non-linear algebraic equations as in (5).

$$\begin{aligned} G_k(z_k, z_{k+1}) = 0, \quad k = 1, 2, \ldots \\ z_k = (x_k, y_k). \end{aligned} \tag{5}$$

The algorithm for calculating $z_k$ is defined by algorithm 1. If $z_{k+1}$ can be explicitly calculated in every step:

$$z_{k+1} = H_k(z_k),$$

we refer to an explicit numerical method. Otherwise, we refer to an implicit method which is generally more accurate.

$z_0 = (x_0, y_0)$;
**while** *requirement is not met* **do**
  Find null-point $z_{k+1}$ of equation $G_k(z_k, z_{k+1}) = 0$;
  $k \leftarrow k + 1$;
**end**

**Algorithm 1:** Calculation of the series $(z_k)_k$

## Exercise 3

**Provide formulas for a series of algebraic equations as in (5) for the system of differential equations (4). To paraphrase, determine functions $G_k(z_k, z_{k+1}) = F_k(x_k, y_k, x_{k+1}, y_{k+1})$ to calculate null-points. It is necessary to provide both an explicit and implicit method.**

Solutions to the exercises are to be submitted on a piece of paper. The procedure has to be legible and clearly formulated and all claims elaborated on.

# Day 2 - Mathematical aircraft

## Introduction

If you have not been run over by vicious STEMartians, come aboard. You will have to demonstrate your programming and management skills in unbearable conditions. Remember, the entire group of STEManiacs depend on you so do not disappoint them.

As experiences professionals, you will use Git for downloading and uploading your code. Every team has an identical initial repository. Two files await you:

- `Drive.m`,
- `LoadScenario.m`,

and folder `simulatorcases`.

# Implementation of numerical solving of differential equations

## 2.1 Implementation of Simulator

You need to implement a numerical method developed during the first day and test it with different test examples.

`Drive` is a basic testing function. It is used, among other things, to call for function `LoadScenario` and `Visualize`. `LoadScenario` is the function which completes starting launching conditions, `BoostPlan` table and other constants required for flight modelling. It also contains four functions, namely `RhoRain`, `RhoSand`, `WindVelocity` and `Ground` , which represent weathering conditions on the map.

The examples of calling for functions are as follows:

```
» x = [40; 50];
» Scenario.RhoSand(x)
ans = 41.875
```

Syntax `structure.element` is used for structure elements..

The aim of the second day is to calculate trajectory $(x_k)_k$ and speed $(y_k)_k$ for the situations defined in the folder `cases`. Series $(x_k)_k$ and $(y_k)_k$ are saved as $2 \times N$ matrices with $i$-column representing vector $x_i \in \mathbb{R}^2$, i.e. $y_i \in \mathbb{R}^2$.

The function `Drive` Drive has a discretization step Timestep defined. The step has a fixed value of 0.01 in all simulations. Drive calls for function Simulator which calculates the aircraft's trajectory. Simulator provides all pieces of information and iteratively calculates the aircraft's trajectory. The trajectory is calculated until the requirements for stopping are met. The requirements for stopping can be the aircraft leaving the trajectory, crashing on the ground, etc. A solution to a non-linear algebraic equation system is searched for in each iteration. Such systems can be solved with Newton method for non-linear equation systems.

Pursuant to `Drive` function, `f`unction, whose purpose is to visualize the entire situation on the map and the aircraft's trajectory, is called for. The folder cases contains testing scenario examples used to test a developed model.

# Exercise

Implement function `Simulator`. `Simulator` has to provide matrix x in format 2 x N in order to visualize it with `Visualize` function, which also needs to be implemented (Note: visualization needs not to be done in .m script). Simulator has to receive structure `Scenario` and discretization step `TimeStep`. Implicit method implementation is more valuable than explicit method implementation. In case of an implicit method, implementation of a method for solving non-linear equations is evaluated. Test your code on the examples for `simulatorcases` folder.

## Downloading and submitting a solution

In order to submit your solutions, use Git system.

# Downloading and submitting a solution

## 2.2 Code downloading

Every team has a git repository on the official STEM Games GitHub. Competitors can access the repository by using their GitHub profiles. If you do not have a profile, create it. The access is granted by sending a message `<TeamName> - <UserName>` (For example, `STEMentors - gospodnetic`) on Slack.

To access the code, you need to install some of git clients and create a `git clone <URL-repository>`.

## 2.3 Code submission

A solution is submitted by using git push command. Make sure that you used all changes in Git system by using `git add` and `git commit` commands.

A submitted code has to be legible and

- be formatted in a clear manner;
- has consistent conventions of line indents;
- has variables' names described.

# Day 3 - Mathematical aircraft

## Introduction

If you had fun so far, we'll make your life more complicated. You'll tell your story if you are lucky enough to survive it. We'll know the finalists today so gear up and show us what you got. Launch the aircraft straight into the STEMars orbit.

## Implementation of the method for flying the aircraft

You need to implement the method for flying the aircraft (boost plan) for the purpose of achieving the desirable aircraft's trajectory.

**Preparation:** Download new data files from GitHub by using `git pull` command. You will find three data files, namely:

- `Drive.m`
- `LoadScenario.m`
- `Visualize`

and folder travelcases.

The same as yesterday, `Drive` is used for testing. Functions `LoadScenario` and `Visualize` upload and visualize the situation on the map. `LoadScenario` uploads data in structure `Scenario` which has new data called *trail*. *Trail* is a series of points on the map we would like the aircraft to visit during its flight. This way we can bypass acid rain and sandy clouds which would slow the aircraft down. Trail is defined in the scenarios found in folder `travelcases` as 2 x M matrix with M being the number of visiting points. A point is visited if we passed it in the radius less than R with R being the predefined constant.

In `Drive`, we call for function `Travel` which calculates the aircraft's trajectory pursuant to the instructions from matrix *Trail*. When calculating the trajectory, you should use the previously developed `Simulator`. `Travel` flies the aircraft in a way to generate its table *boost plan*.

Inputs in *boost plan* are generated sequentially so each input guides the aircraft to the next destination. For each input, choose the turn of a booster and booster force intensity. Since the aircraft's destination depends on starting intensity, it is possible to use search implementation by running the bisection method. When choosing the turn and intensity, take the used fuel into account because the aircraft cannot fly without this resource.

All needed lines of the code from day 2 copy into Day3 folder.

**Note:** The official visualization function needs not to be used.

# Exercise

Implement `Travel` function. Travel needs to return matrix x and boost plan used by the aircraft to visit the points defined in matrix trail. Travel needs to receive structure `Scenario` and discretization step `TimeStep`. The allowed distance from the target is R = 10.

Test your code in the cases from folder `travecases`.

## Code submission

A solution is submitted by using `git push` command. Make sure that you used all changes in Git system by using `git add` and `git commit` commands.

A submitted code has to be legible and

- be formatted in a clear manner;
- has consistent conventions of line indents;
- has variables' names described.