1. Theory

In this section the theory behind the implementation of the MATLAB code will be given. Figure 1 displays the bucket and the pile of gravel. We assume that the end point of the bucket follow the curve $\Gamma(t) = (x(t), y(t))$ and that the angle β follow the function $\beta(t)$ (where t is the parametrization variable). We also assume an infinite pile of gravel.

g	Gravitational acceleration
ρ_{gravel}	The density of the gravel

Table 1

Potential energy

The potential energy is the change of potential energy between the initial and the final state of the bucket and the gravel. The potential energies of the initial state of the bucket and the final state of the bucket and gravel are constant. Hence, from a optimization point of view, the only potential energy that will be interesting is the one for the initial state of the gravel. It is determined by the following integral

$$E_P = g \int y \cdot dm = g \cdot \rho_{\text{gravel}} \int_{\Gamma(t)} y(t) \left(\frac{y(t)}{\tan \gamma} - x(t) \right) dy(t)$$

Compression energy

A compression force is assumed to always be present and is pointing in the same direction as the lower part of the bucket (this may not be the correct assumption but it seems to be close enough). The compression force is determined using Bekker's relation

$$F_C(t) = \left(\frac{k_C}{b} + k_\varphi \right) \left(\frac{y(t)}{\tan \gamma} - x(t) \right)^n = k_{\text{tot}} \left(\frac{y(t)}{\tan \gamma} - x(t) \right)^n$$

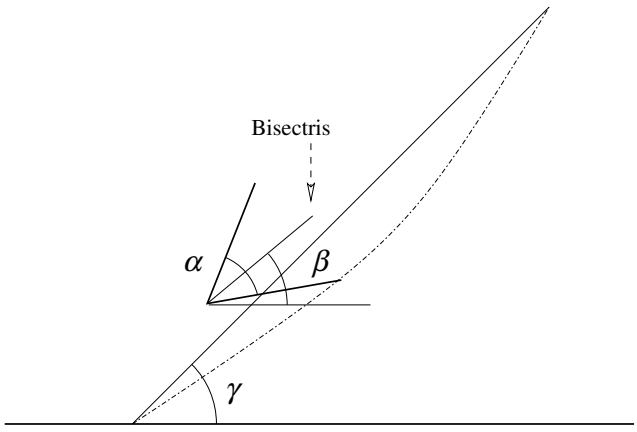

Figure 1 A schematic view of the bucket and the pile of gravel. The angle α is the angle of the rear of the bucket. The angle β is the angle between a horizontal line and the bisectris of the bucket. The angle γ is the angle of the pile of gravel.

Hence the energy arising from the compression force will be

$$E_C = \int_{\Gamma(t)} F_C(t) d\boldsymbol{s} =$$
$$k_{\text{tot}} \int_{\Gamma(t)} \left(\frac{y(t)}{\tan\gamma} - x(t) \right)^n \left(\cos\left(\beta(t) - \frac{\alpha}{2}\right) dx(t) + \sin\left(\beta(t) - \frac{\alpha}{2}\right) dy(t) \right)$$

Discretization of the integrals

The total energy required to lift the bucket and the gravel from the initial state to the final state will be

$$E = dE_{\text{potential}} + E_C = [constant] - E_P + E_C$$

(where $dE_{\text{potential}}$ denotes the change in potential energy). We would like to find the minimal value of E under the constraint that the bucket is full in its final state, that is the volume of the gravel within $\Gamma(t)$ equals the bucket volume. To minimize over the integrals given in the previous parts, we will discretize the trajectory $\Gamma(t)$. The discretized trajectories becomes $x(t) = \{x_0 = 0, x_1, \ldots, x_{N-1}, x_N = y_{\max}/\tan(\pi/2)\}$, $y(t) = \{y_0 = 0, y_1, \ldots, y_{N-1}, y_N = y_{\max}\}$ and $\beta(t) = \{\beta_0 = \frac{\alpha}{2}, \beta_1, \ldots, \beta_{N-1}, \beta_N = \frac{\pi}{2}\}$. But we will choose the discretization of $y(t)$ such that $y_{k+1} - y_k = \Delta y$ for all k. Hence the values of y_k are constant. The decision variables of the optimization program will only be $x_1, x_2, \ldots x_{N-1}$ and $\beta_1, \beta_2, \ldots \beta_{N-1}$. Approximating the integral using the trapezoidal rule and letting $dx_k = \frac{x_{k+1} - x_{k-1}}{2}$, we get

$$E_P \approx g \cdot \rho_{\text{gravel}} \sum_{k=1}^{N-1} y_k \left(\frac{y_k}{\tan(\alpha/2)} - x_k \right) \Delta y \tag{1}$$

$$E_C \approx k_{\text{tot}} \sum_{k=1}^{N-1} \left(\frac{y_k}{\tan(\alpha/2)} - x_k \right)^n \left(\cos\left(\beta_k - \frac{\alpha}{2}\right) \frac{x_{k+1} - x_{k-1}}{2} + \sin\left(\beta_k - \frac{\alpha}{2}\right) \Delta y \right) \tag{2}$$

2. Implementation

A description of the MATLAB code will be given in this section. Only the callback functions will be described. Before the description of the functions, a table of common variable names used in the functions will be given

objective

In this function (1) and (2) are implemented. The implementation is straight forward.

constraints and constraintsLimits

In these functions the constraints are realized. There are in total 4 different kinds of constraints. These are

1. Gravel volume - bucket volume $= 0$, meaning that $0 \leq \sum X_k \Delta y \leq 0$.
2. This constraint says that $-d\beta_{\max} \leq \beta_{k+1} - \beta_k \leq d\beta_{\max}$. This constraint is used to avoid the optimization routine to choose a β trajectory that changes too fast. In the code $d\beta_{\max}$ is the parameter `bucket.maxDTiltAngle`.

`xIn`	All decision variables (both x and β)
`x`	The decision variables $x_1, \ldots, x_{N-1}$
`bucketTiltAngle`	The decision variables $\beta_1, \ldots, \beta_{N-1}$
`y`	The constants $y_0, \ldots, y_N$
`b`	Equals $\beta - \alpha/2$
`X`	Equals $\frac{y}{\tan(\beta-\alpha/2)} - x$
`Y`	Denotes $y_1, \ldots, y_{N-1}$
`s`	The side length of the bucket

Table 2

3. This constraint says that the rear of the bucket is not allowed below the ground. Hence $0 \leq y_k - s \cdot \sin(\beta_k - \alpha/2) < \infty$ for $k = 1, \ldots, N-1$.
4. This constraint says that the x trajectory should be increasing, that is $0 \leq x_{k+1} - x_k < \infty$.

Also, `constraintsLimits` contain lower and upper bounds on the decision variables, x and β. They are $\frac{y}{\tan\gamma} \leq x \leq \frac{y_{\max}}{\tan\gamma}$ and $\frac{\alpha}{2} \leq \beta \leq \frac{\pi}{2}$ and are found in `limits.lb` and `limits.ub`, respectively.

gradient

The function should return the gradient of the cost function (that is `objective`). The gradient is of course the sum of the gradients of (1) and (2). The gradient of (1) is quite simple, but the gradient of (2) is more complicated.

jacobian and jacobianstructure

The function `jacobian` should return the jacobian of the constraints as a sparse matrix. That is, if we call the returned value of `jacobian` for J and the returned value of `constraints` for C, we should have that

$$J = \begin{bmatrix} \nabla C_1^T \\ \nabla C_2^T \\ \vdots \\ \nabla C_s^T \end{bmatrix}$$

assuming that there are s constraints.

The function `jacobianstructure` should return the structure of `jacobian` as a sparse matrix. In essence,

```
jacobianstructure = (jacobian ~= 0)
```

But this implementation should not be used for obvious reasons.

hessian and hessianstructure

The function `hessian` should return a weighted hessian of the cost function and each hessian of the constraints. For the inputs `sigma` and `lambda` to `hessian` we should have

```
hessian = sigma*H(objective) + sum(lambda(k)*H(constraints(k)))
```

The function `hessianstructure` should work similar as `jacobianstructure`.

3. What to do next

There are a few things that could be the next step in this approach

- Change parametrization of $\Gamma(t)$.
- Make sure that the rear of the bucket does not go into the pile of gravel at any point of the trajectory.
- Add more forces (for example breaking force) in order to make the energy more realistic.
- Limitations/performance of the hydraulics etc when moving the bucket.