Problem Formulation

```
function [n,m] = Hw5_Initialize(dt)
```

The following function auto-generates all the necessary functions for rewriting the Nonlinear Program (NLP)

$$\begin{aligned} &\min \; J_N(x_N) + \sum_{k=0}^{N-1} J_k(x_k, u_k) \\ &\text{s.t.} \;\; x_{k+1} = f(x_k, u_k), \qquad k \in [0, N-1], \quad x_0 = x \end{aligned}$$

as the Sequential Quadratic Progam (SQP)

$$\min \; \frac{1}{2} \Delta x_N^T \, P(x_N) \, \Delta x_N + \Delta x_N^T \, p(x_N) + \sum_{k=0}^{N-1} \frac{1}{2} \begin{bmatrix} \Delta x_0 \\ \Delta u_0 \end{bmatrix}^T \begin{bmatrix} Q(x_k, u_k, \lambda_k) & S(x_k, u_k, \lambda_k)^T \\ S(x_k, u_k, \lambda_k) & R(x_k, u_k, \lambda_k) \end{bmatrix} \begin{bmatrix} \Delta x_0 \\ \Delta u_0 \end{bmatrix} + \begin{bmatrix} \Delta x_0 \\ \Delta u_0 \end{bmatrix}^T \begin{bmatrix} q(x_k, u_k) \\ r(x_k, u_k) \end{bmatrix}$$

$$\text{s.t.} \;\; A(x_k, u_k)\Delta x_k + B(x_k, u_k)\Delta u_k - \Delta x_{k+1} = x_{k+1} - f(x_k, u_k), \qquad k \in [0, N-1], \quad \Delta x_0 = 0,$$

which features the following linear-quadratic approximations:

- **Jacobian of the System Dynamics**

$$A(x, u) = \nabla_x f(x, u), \quad B(x, u) = \nabla_u f(x, u),$$

- **Gradient of the Cost Function**

$$p(x) = \nabla J_N(x), \quad q(x, u) = \nabla_x J_k(x, u), \quad r(x, u) = \nabla_u J_k(x, u)$$

- **Hessian of the Lagrangian**

$$P_1(x) = \nabla^2 L_N(x), \quad Q_1(x, u, \lambda) = \nabla^2_{xx} L_k(x, u, \lambda), \quad R_1(x, u) = \nabla^2_{uu} L_k(x, u, \lambda), \quad S_1(x, u) = \nabla^2_{ux} L_k(x, u, \lambda)$$

given $L_N(x) = J_N(x)$ and $L_k(x, u, \lambda) = J_k(x, u) + \lambda^T f(x, u)$

- [...] or **Hessian of the Cost Function**

$$P_2(x) = \nabla^2 J_N(x), \quad Q_2(x, u) = \nabla^2_{xx} J_k(x, u), \quad R_2(x, u) = \nabla^2_{uu} J_k(x, u), \quad S_2(x, u) = \nabla^2_{ux} J_k(x, u)$$

just in case the Hessian of the Lagrangian is not positive definite.

Note that, since $L_N(x) = J_N(x)$, we have that $P_1(x) = P_2(x)$, making the distinction between these two definitions irrelevant.

Problem Size

We begin by defing all our symbolic variables $x \in R^n$, $u \in R^m$, $\lambda \in R^n$.

```
% System Size
n = 6;
m = 2;

% Refernece
ref = [5; 0; 0; 0; 0; 0];

% Define symbolic variables
l = sym('l',[n 1]);
x = sym('x',[n 1]);
u = sym('u',[m 1]);
```

System Dynamics

We now define the dynamic model of our specific system. Here, we consider the nonlinear bycicle model

$$\begin{bmatrix} \dot{y} \\ \dot{v} \\ \dot{\varphi} \\ \dot{\omega} \\ \dot{\delta}_f \\ \dot{\delta}_r \end{bmatrix} = \begin{bmatrix} s\sin(\varphi) + v\cos(\varphi) \\ -s\omega + \dfrac{1}{M}(F(\alpha_f)\cos(\delta_f) + F(\alpha_r)\cos(\delta_r)) \\ \omega \\ \dfrac{1}{J}\left(F(\alpha_f)\cos(\delta_f)L_f - F(\alpha_r)\cos(\delta_r)L_r\right) \\ u_1 \\ u_2 \end{bmatrix},$$

where

$$\alpha_f = \delta_f - \mathrm{atan}\left(\frac{v + L_f\,\omega}{s}\right)$$

$$\alpha_r = \delta_r - \mathrm{atan}\left(\frac{v - L_r\,\omega}{s}\right)$$

are the sideslip angles of the front and rear tires, respectively,

$$F(\alpha) = \mu\, m\, g \sin(c\, \mathrm{atan}(b\alpha))$$

is the Pacejka model for tire forces, and all other variables are positive constants.

```
% Model parameters
M = 2041;   % [kg]     Vehicle mass
I = 4964;   % [kg m^2] Vehicle inertia (yaw)
Lf = 1.56;  % [m]      CG distance, front
Lr = 1.64;  % [m]      CG distance, back
mu = 0.8;   % []       Coefficient of friction
b = 12;     % []       Tire parameter (Pacejka model)
c = 1.285;  % []       Tire parameter (Pacejka model)
s = 30;     % [m/s]    Vehicle speed
g = 9.81;   % [m/s^2]  Gravity acceleration
```

```
% Sideslip angles
af = x(5)-atan((x(2)+Lf*x(4))/s);
ar = x(6)-atan((x(2)-Lr*x(4))/s);

% Tire forces
Ff = mu*g*M*sin(c*atan(b*(af)));
Fr = mu*g*M*sin(c*atan(b*(ar)));

% Dynamic Model (continuos-time)
fc = [ s*sin(x(3))+x(2)*cos(x(3))
      -s*x(4)+(Ff*cos(x(5))+Fr*cos(x(6)))/M
       x(4)
      (Ff*cos(x(5))*Lf-Fr*cos(x(6))*Lr)/I
       u(1)
       u(2)];

Constraint = [ -x(5)-deg2rad(30)
                x(5)-deg2rad(30)
               -x(6)-deg2rad(6)
                x(6)-deg2rad(6)
               -u(1)-1
                u(1)-1
               -u(2)-1
                u(2)-1];
               % -af-deg2rad(6)
               %  af-deg2rad(6)
               % -ar-deg2rad(6)
               %  ar-deg2rad(6)];
```

Discrete-time Approximation

We now define the discrete-time dynamic model $x_{k+1} = f(x_k, u_k)$ by taking the Forward-Euler approximation

```
% Forward Euler approximation
f = x + dt*fc;
```

Jacobian of the System Dynamics

After computing $A(x, u) = \nabla_x f(x, u)$ and $B(x, u) = \nabla_u f(x, u)$ symbolically, we save them as the auto-generated MATLAB functions **`A.m`** and **`B.m`**.

```
% Jacobians
Asym = jacobian(f,x);
Bsym = jacobian(f,u);

% Convert symbolic functions to MATLAB functions
matlabFunction(f,'File','f','Vars',{x,u});
```

```
matlabFunction(Asym,'File','A','Vars',{x,u});
matlabFunction(Bsym,'File','B','Vars',{x,u});
matlabFunction(Constraint,'File','Constraint','Vars',{x,u});
```

Cost Function

For simplicity, we define the quadratic cost functions

$$J_N(x) = \frac{1}{2}x^T P\, x \quad \text{and} \quad J_k(x,u) = \frac{1}{2}x^T Q\, x + \frac{1}{2}\, u^T\, R\, u,$$

where Q and R are identity matrices. To obtain P, we will linearize the system around the origin and then solve the Discrete Algebraic Riccati Equation.

```
% Stage Cost
Qq = diag([1 0 1 0 0 0]);
Rr = 0.1*eye(2);

J_k = 1/2*(x-ref)'*Qq*(x-ref)+1/2*u'*Rr*u;

% Terminal Cost
[~,Pp,~] = dlqr(A(zeros(6,1),0),B(zeros(6,1),0),Qq,Rr);

J_N = 1/2*(x-ref)'*Pp*(x-ref);
% J_N = 0;
```

Gradient of the Cost Function

After computing $p(x) = \nabla J_N(x)$, $q(x,u) = \nabla_x J_k(x,u)$, and $r(x,u) = \nabla_u J_k(x,u)$ symbolically, we save them as the auto-generated MATLAB functions **p.m**, **q.m** and **r.m**.

```
% Gradients
q = jacobian(J_k,x);
r = jacobian(J_k,u);
p = jacobian(J_N,x);
c_x = jacobian(Constraint,x);
c_u = jacobian(Constraint,u);

% Convert symbolic functions to MATLAB functions
matlabFunction(q,'File','q','Vars',{x,u});
matlabFunction(r,'File','r','Vars',{x,u});
matlabFunction(p,'File','p','Vars',{x,u});
matlabFunction(c_x,'File','c_x','Vars',{x,u});
matlabFunction(c_u,'File','c_u','Vars',{x,u});
```

Hessian of the Cost Function

We can do the same for the Hessian of the cost function, i.e. $P_L(x) = \nabla^2 J_N(x)$, $Q_J(x,u) = \nabla^2_{xx} J_k(x,u)$, $R_J(x,u) = \nabla^2_{uu} J_k(x,u)$, and $S_J(x,u) = \nabla^2_{ux} J_k(x,u)$, which we save as the auto-generated MATLAB functions **`PN.m`**, **`QG.m`**, **`RG.m`**, and **`SG.m`**.

```
% Hessians (of the Cost)
Q = jacobian(q,x);
R = jacobian(r,u);
S = jacobian(r,x);
P = jacobian(p,x);

% Convert symbolic functions to MATLAB functions
matlabFunction(Q,'File','QJ','Vars',{x,u});
matlabFunction(R,'File','RJ','Vars',{x,u});
matlabFunction(S,'File','SJ','Vars',{x,u});
matlabFunction(P,'File','PN','Vars',{x,u});
```

Hessian of the Lagrangian

Finally, we compute the Hessian of the Lagrangian, i.e. $P_L(x) = \nabla^2 L_N(x)$, $Q_L(x,u,\lambda) = \nabla^2_{xx} L_k(x,u,\lambda)$, $R_L(x,u) = \nabla^2_{uu} L_k(x,u,\lambda)$, and $S_L(x,u,\lambda) = \nabla^2_{ux} L_k(x,u,\lambda)$, which we save as the auto-generated MATLAB functions **`QL.m`**, **`RL.m`**, and **`SL.m`**. [Note: $P_L(x)$ has already been saved as **`PN.m`**].

```
%% Lagrangian
L_k = J_k + l'*f;

% Hessians (of the Lagrangian)
Q = jacobian(jacobian(L_k,x),x);
R = jacobian(jacobian(L_k,u),u);
S = jacobian(jacobian(L_k,u),x);

% Convert symbolic functions to MATLAB functions
matlabFunction(Q,'File','QL','Vars',{x,u,l});
matlabFunction(R,'File','RL','Vars',{x,u,l});
matlabFunction(S,'File','SL','Vars',{x,u,l});

end
```