Generate sub-QP

```
function [H,h,E,e,Ec,c] =  gen_QP(z,l,x0,n,m,N,cn)
```

This function generates the matrices for the Quadratic Program

$$
\begin{aligned}
&\min \quad \frac{1}{2}\Delta z^T\, H\, \Delta z + \Delta z^T h \\
&\text{s.t.} \quad E\, \Delta z = e
\end{aligned}
$$

where, given

$$
\Delta z = \begin{bmatrix} \Delta u_0 \\ \Delta u_1 \\ \vdots \\ \Delta u_{N-1} \\ \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_N \end{bmatrix}
$$

we wish to solve the optimal control problem

$$
\min \ \frac{1}{2}\Delta x_N^T\, P_N\, \Delta x_N + \Delta x_N^T\, p_N + \sum_{k=0}^{N-1} \frac{1}{2}\begin{bmatrix} \Delta x_0 \\ \Delta u_0 \end{bmatrix}^T \begin{bmatrix} Q_k & S_k^T \\ S_k & R_k \end{bmatrix} \begin{bmatrix} \Delta x_0 \\ \Delta u_0 \end{bmatrix} + \begin{bmatrix} \Delta x_0 \\ \Delta u_0 \end{bmatrix}^T \begin{bmatrix} q_k \\ r_k \end{bmatrix}
$$

$$
\text{s.t.} \ \ A_k \Delta x_k - B_k \Delta u_k - \Delta x_{k+1} = x_{k+1} - f_k, \qquad\qquad k \in [0, N-1], \quad \Delta x_0 = 0.
$$

Trajectory Extraction

The first step in generating the QP matrices is to use the vector

$$
z = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix},
$$

to compute u_0 and generate the two matrices

$$X = \begin{bmatrix} x_1 & \dots & x_{N-1} & x_N \end{bmatrix}$$
$$U = \begin{bmatrix} u_1 & \dots & u_{N-1} \end{bmatrix}$$

```
% Extract state and input trajectories from z
U = reshape(z(    1:N*m),m,N);
X = reshape(z(N*m+1:end),n,N);
L = reshape(l           ,n,N);

u0 = U(1:m,1);
U = U(:,2:end);
```

Matrix Creation

After initializing the matrices

```
% Initialization
HqL = zeros(n*N);       HqJ=HqL;
HrL = zeros(m*N);       HrJ=HrL;
HsL = zeros(m*N,n*N);   HsJ=HsL;
hr = zeros(m*N,1);
hq = zeros(n*N,1);

Ea =-eye(n*N);
Eb = zeros(n*N,m*N);
 e = zeros(n*N,1);

Ec = zeros(cn*N,n*N);
Ed = zeros(cn*N,m*N);
 c = zeros(cn*N,1);
```

we wish to generate H, h, E, e

$$
H = \begin{bmatrix} R_0 & & \\ & \begin{bmatrix} \underbrace{\begin{bmatrix} R_1 & & \\ & \ddots & \\ & & R_{N-1} \end{bmatrix}}_{H_r} & \begin{bmatrix} S_1^T & & \\ & \ddots & \\ & & S_{N-1}^T \end{bmatrix} \\ \underbrace{\begin{bmatrix} S_1 & & \\ & \ddots & \\ & & S_{N-1} \end{bmatrix}}_{H_s} & \underbrace{\begin{bmatrix} Q_1 & & \\ & \ddots & \\ & & Q_{N-1} \end{bmatrix}}_{H_q} \end{bmatrix} & \\ & & P_N \end{bmatrix}, \qquad h = \begin{bmatrix} r_0 \\ \begin{bmatrix} r_1 \\ \vdots \\ r_{N-1} \end{bmatrix} \\ \begin{bmatrix} q_1 \\ \vdots \\ q_{N-1} \end{bmatrix} \\ p_N \end{bmatrix}
$$

$$
E = \begin{bmatrix} \underbrace{\begin{bmatrix} B_0 & & & \\ & B_1 & & \\ & & \ddots & \\ & & & B_{N-1} \end{bmatrix}}_{E_b} & \underbrace{\begin{bmatrix} -I & & & \\ A_1 & -I & & \\ & \ddots & -I & \\ & & A_{N-1} & -I \end{bmatrix}}_{E_a} \end{bmatrix}, \qquad e = \begin{bmatrix} x_1 - f_0 \\ x_2 - f_1 \\ \vdots \\ x_N - f_{N-1} \end{bmatrix},
$$

where $f_k = f(x_k, u_k)$, $A_k = A(x_k, u_k)$, $B_k = B(x_k, u_k)$, $p_k = p(x_k)$, $q_k = q(x_k, u_k)$, $r_k = r(x_k, u_k)$, $P_N = P(x_N)$.

Since the second-order terms Q_k, R_k, S_k can be obtained using either the **Hessian of the Lagrangian** or the **Hessian of the Cost Function**, we compute two different versions of the matrix H, i.e. H_L and H_J.

```
% Update elements corresponding to k = 0
HrL(1:m,1:m) = RL(x0,u0,L(:,1));
HrJ(1:m,1:m) = RJ(x0,u0);

hr(1:m,1  ) =  r(x0,u0);

Eb(1:n,1:m)   = B(x0,u0);
 e(1:n,1  )   = X(:,1)-f(x0,u0);

Ed(1:cn,1:m) = c_u(x0,u0);
 c(1:cn,1 )   = -Constraint(x0,u0);

% Update all elements from k=1 to k=N-1
for k = 1:N-1
    % Update elements corresponding to current k
    HrL( k   *m+(1:m), k   *m+(1:m)) = RL(X(:,k),U(:,k),L(:,k+1));
    HsL( k   *m+(1:m),(k-1)*n+(1:n)) = SL(X(:,k),U(:,k),L(:,k+1));
    HqL((k-1)*n+(1:n),(k-1)*n+(1:n)) = QL(X(:,k),U(:,k),L(:,k+1));

    HrJ( k   *m+(1:m), k   *m+(1:m)) = RJ(X(:,k),U(:,k));
    HsJ( k   *m+(1:m),(k-1)*n+(1:n)) = SJ(X(:,k),U(:,k));
    HqJ((k-1)*n+(1:n),(k-1)*n+(1:n)) = QJ(X(:,k),U(:,k));

    hr( k   *m+(1:m),1            ) =  r(X(:,k),U(:,k));
```

```
    hq((k-1)*n+(1:n),1            ) =  q(X(:,k),U(:,k));

    Ea(k*n+(1:n),(k-1)*n+(1:n)) = A(X(:,k),U(:,k));
    Eb(k*n+(1:n), k   *m+(1:m)) = B(X(:,k),U(:,k));
     e(k*n+(1:n), 1)            = X(:,k+1)-f(X(:,k),U(:,k));
    Ec(k*cn+(1:cn),(k-1)*n+(1:n))  = c_x(X(:,k),U(:,k));
    Ed(k*cn+(1:cn), k   *m+(1:m))  = c_u(X(:,k),U(:,k));
     c(k*cn+(1:cn), 1)               = -Constraint(X(:,k),U(:,k));
end

% Update elements corresponding to k = N
HqL((N-1)*n+(1:n),(N-1)*n+(1:n)) = PN(X(:,N));
HqJ((N-1)*n+(1:n),(N-1)*n+(1:n)) = PN(X(:,N));

hq((N-1)*n+(1:n),1            ) =  p(X(:,N));

% Compose Matrices
h = [hr
     hq];

E = [Eb Ea];
Ec = [Ed Ec];
```

Hessian Selection

Normally, it is preferable to output the Hessian of the Lagrangian $H = H_L$ since it ensures quadratic convergence in proximity of the optimizer.

```
% Hessian of the Lagrangian
H = [HrL  HsL
     HsL' HqL];
```

However, there is no guarantee that $H_L > 0$, which can cause the sub-QP to become ill-defined.

To prevent such issues (assuming that the NLP cost function is convex), we replace H_L with the Hessian of the Cost Function $H = H_J$ whenever $H_L \not> 0$.

```
if min(eig(H))<=0
    % Hessian of the Cost Function
    H = [HrJ  HsJ
         HsJ' HqJ];
end

end
```