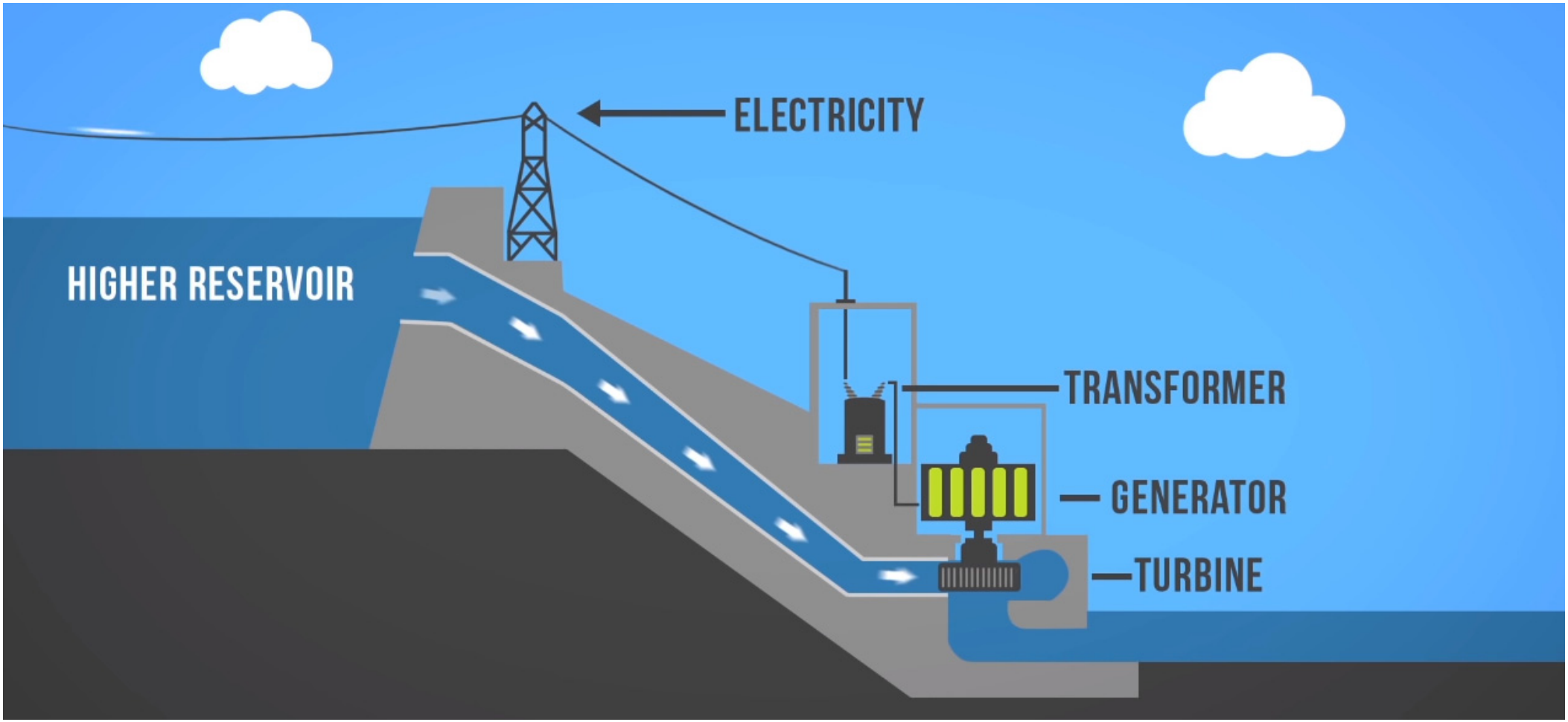
Florida International University
Department of Civil and Environmental Engineering
Optimization in Water Resources Engineering, Spring 2020

LECTURE: OPTIMIZATION OF HYDROPOWER OPERATION

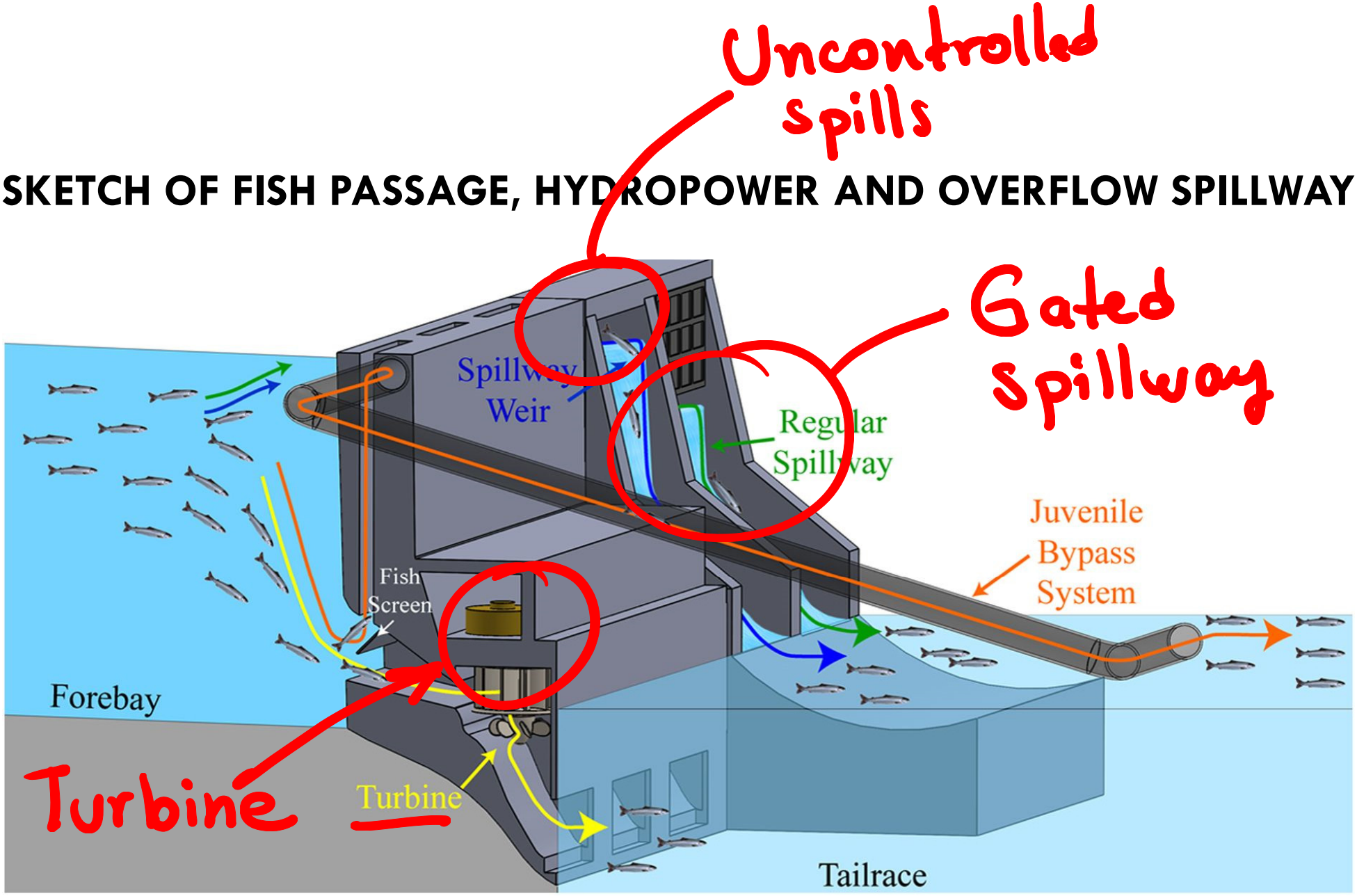
Arturo S. Leon, Ph.D., P.E., D.WRE

BASICS OF HYDROPOWER (VIDEO)



Source: https://www.youtube.com/watch?v=q8HmRLCgDAI

SKETCH OF FISH PASSAGE, HYDROPOWER AND OVERFLOW SPILLWAY

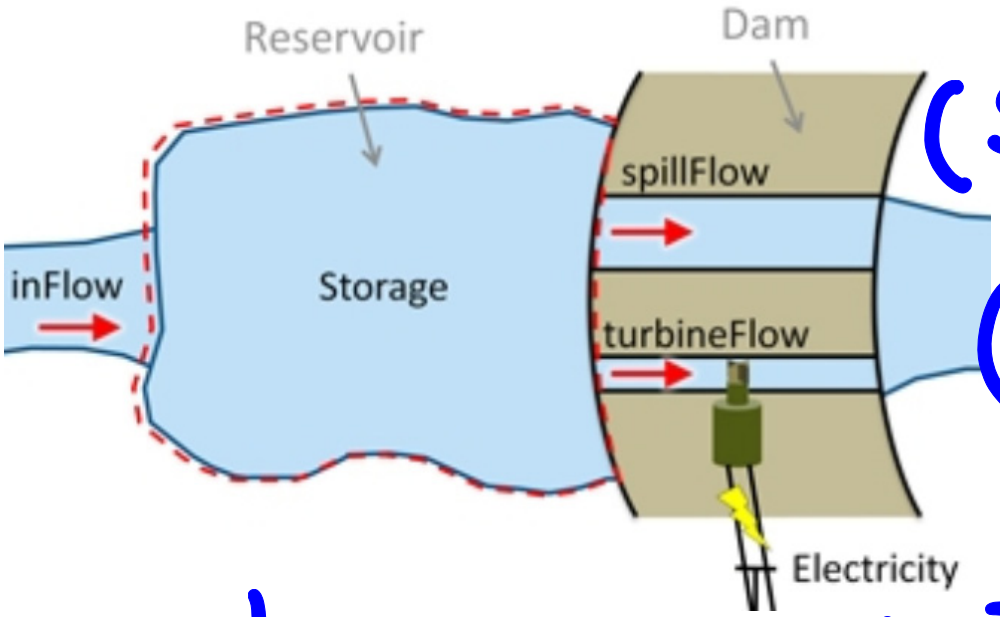


Source: https://www.nature.com/articles/s41598-018-19208-1/figures/1

CONSIDERATIONS:

1. Water can leave the reservoir either through the spillway or via the turbine
2. Water that leaves through the turbine is used to generate electricity and sold at prices determined by the market
3. We control the rates at which water flows through the **turbine** and **gated spillway** (overflow spillway is uncontrolled)
4. Our goal is to find values for the **turbine** and **spillway** flow rates that will maximize revenue over a period of several days at short time-intervals. Herein, we will find the flow values at each hour.

SCHEMATIC OF THE PLAN VIEW OF A HYDROELECTRIC DAM



(SF)

(TF)

$$\text{Total outflow}_t = SF_t + TF_t$$

$$\Sigma I - \Sigma O = \frac{\Delta S}{\Delta t}$$

SYSTEM EQUATIONS

$$E = \eta \rho g Q H_{\text{net}}$$

E = Electricity power in Watts

η = overall hydropower efficiency (~ 70-80%)

ρ = water density (kg/m^3)

g = gravitational constant (9.81 m/s^2)

Q = turbine flow discharge (m^3/s)

H_{net} = net head (m). This is the gross head physically measured at the site, less any head losses. Head losses is around 10-15%. Assuming a head loss of 10%, H_{net}=H_{gross} x 0.9

S = Reservoir storage

t = time

ASSUMPTIONS FOR OUR MODELING

$\eta = 0.7$ (70%)

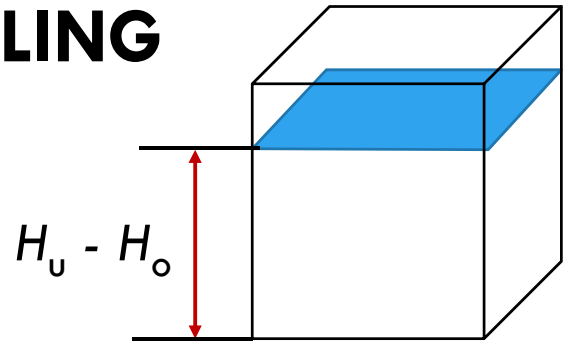
Head losses = 10% H_{gross}

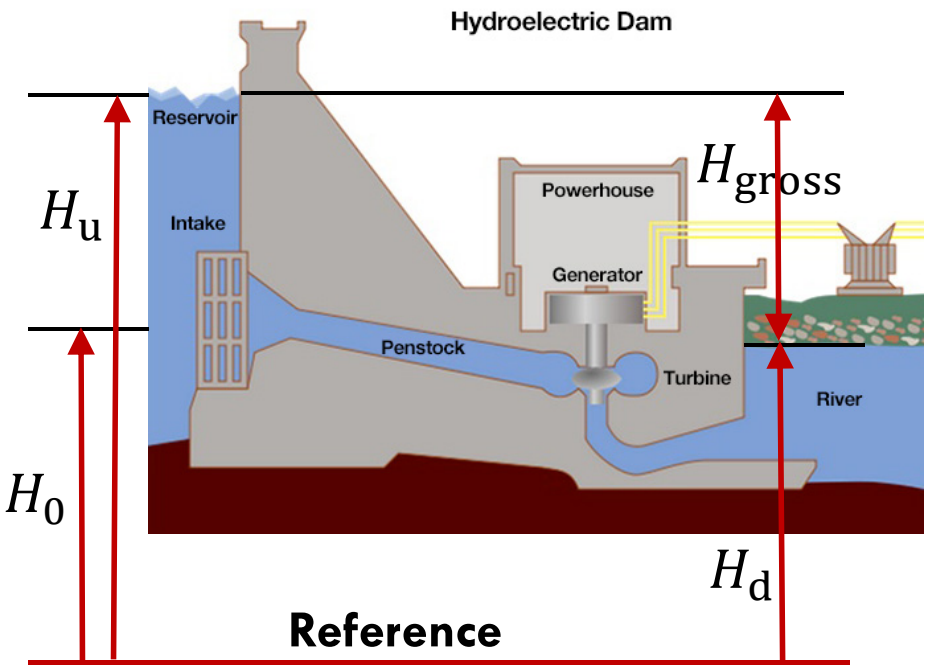
Thus, $H_{net}=H_{gross}$ x 0.9

Then:

$$E = \eta\rho g Q H_{\text{net}} = 6180.3 Q H_{\text{gross}}$$

Also, assume a rectangular reservoir. Thus, $S = A(H_u - H_o)$ where S is the storage of the reservoir above the penstock entrance, A is the surface area of the reservoir, and H_o is the height from a reference level until the centerline of the penstock entrance. Whenever the water level in the reservoir is below the penstock entrance, there is no production of hydropower.





ASSUMPTIONS FOR OUR MODELING (CONT.)

Note that $H_{gross} = H_u - H_d$

Thus, $H_{gross} = S/A + H_o - H_d$

Then:

$$E = 6180.3Q(S/A + H_o - H_d)$$

Note that A and H_o are constants. H_d also remains pretty much constant. Thus,

$$E = Q(k_1 S + k_2)$$

Where:

$k_1 = 6180.3/A$

$k_2 = 6180.3\ (H_o - H_d)$

ASSUMPTIONS FOR OUR MODELING (CONT.)

Assume A = 200 km^2 = 200x10^6 m^2

$H_o - H_d = 1$ m

Then:

$k_1 = 0.0000309015$

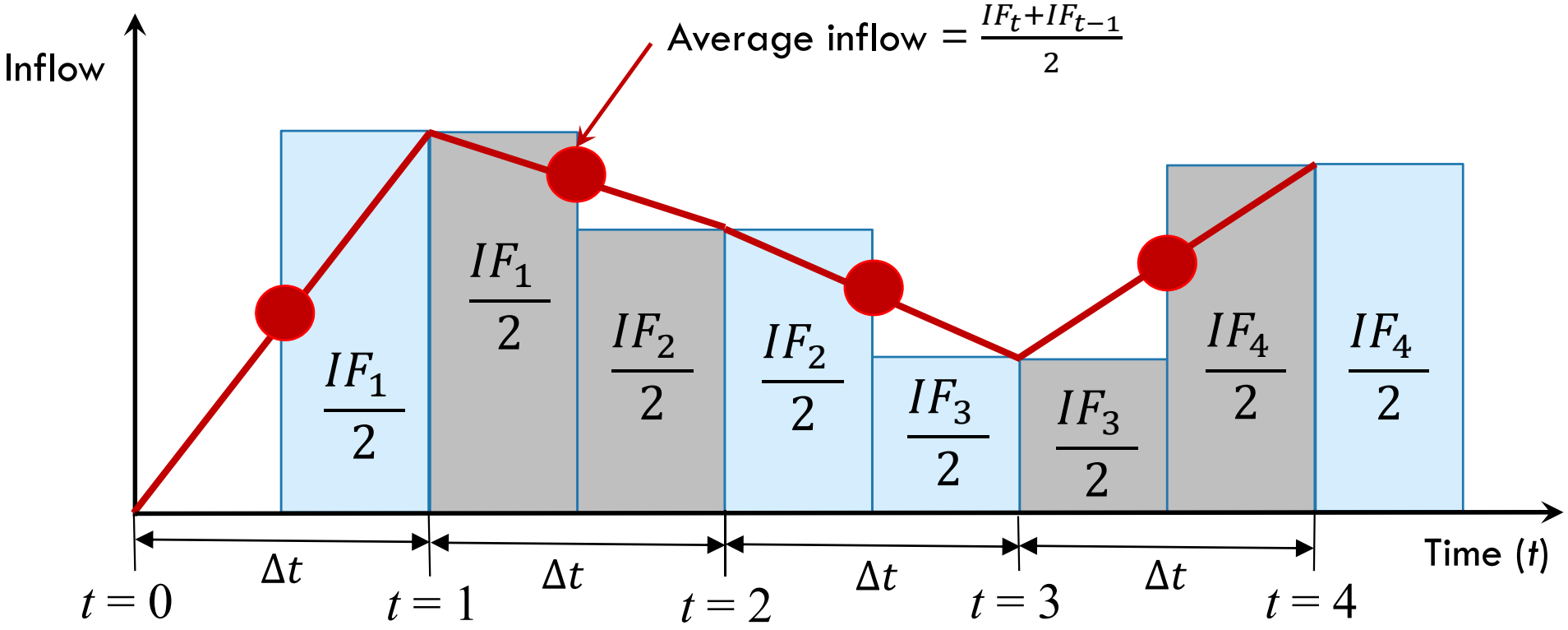
$k_2 = 6180.3$

Discretizing the electricity power:

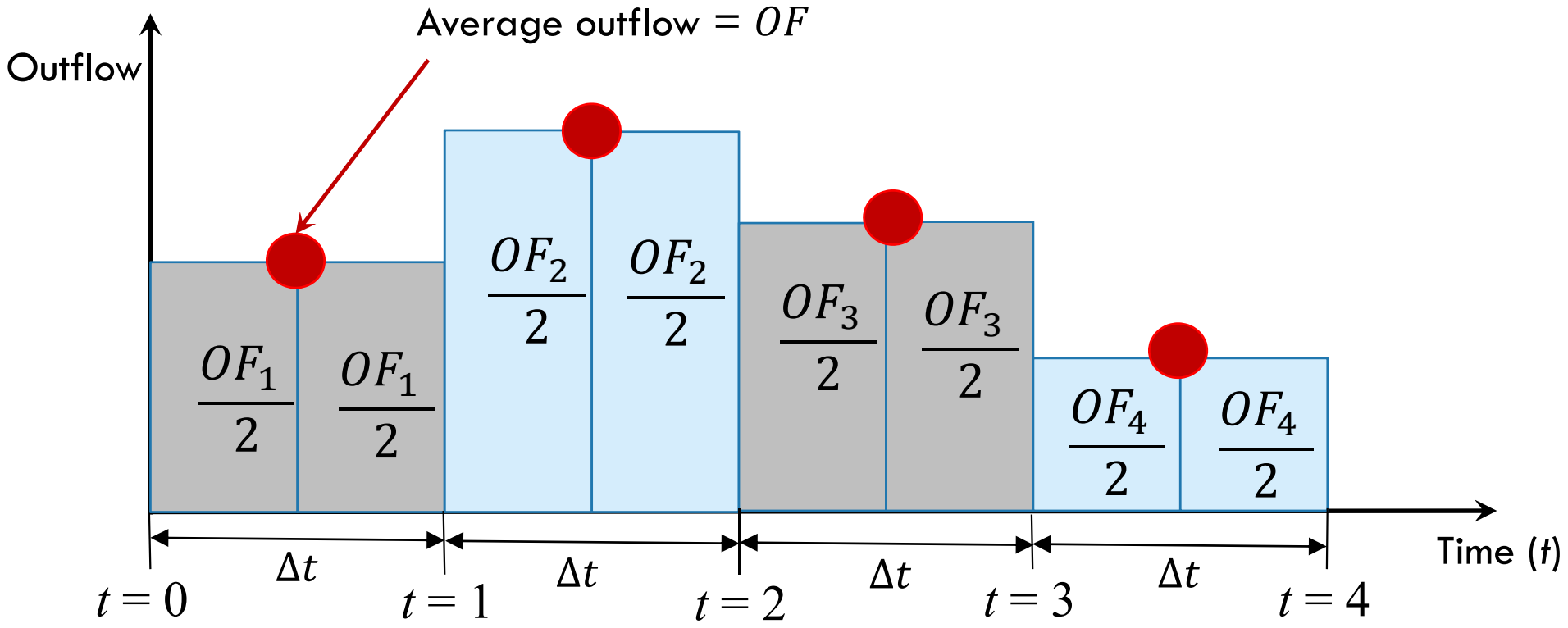
$$E(t) = \text{Turbine flow } (t-1) * [k_1 S(t-1/2) + k_2]$$

Where S(t-1/2) = storage at half time step

STORAGE AT HALF TIME STEP



OUTFLOW (RELEASES ARE DETERMINED AT THE BEGINNING OF TIME STEP



$\Sigma I - \Sigma O = \frac{\Delta S}{\Delta t} \rightarrow S_{t+1} - S_t = (\Sigma I - \Sigma O)\Delta t$

$S_{1/2} = S_0 + \left(\frac{IF_1}{2} - \frac{TF_1}{2} - \frac{SF_1}{2}\right)\Delta t$

STORAGE AT HALF TIME STEP

$$S_{1/2} = S_0 + \Delta t\left[\frac{IF_1}{2} - \frac{TF_1}{2} - \frac{SF_1}{2}\right] = \underbrace{S_0 + \Delta t\left[\frac{IF_1}{2}\right]}_{\text{inloop}_1} - \Delta t\left[\frac{TF_1}{2} + \frac{SF_1}{2}\right]$$

$$S_{3/2} = S_{1/2} + \Delta t\left[\frac{IF_1}{2} + \frac{IF_2}{2} - \left(\frac{TF_1}{2} + \frac{TF_2}{2} + \frac{SF_1}{2} + \frac{SF_2}{2}\right)\right]$$

$$S_{3/2} = \text{inloop}_1 - \Delta t\left[\frac{TF_1}{2} + \frac{SF_1}{2}\right] + \Delta t\left[\frac{IF_1}{2} + \frac{IF_2}{2}\right] - \Delta t\left[\frac{TF_1}{2} + \frac{TF_2}{2} + \frac{SF_1}{2} + \frac{SF_2}{2}\right]$$

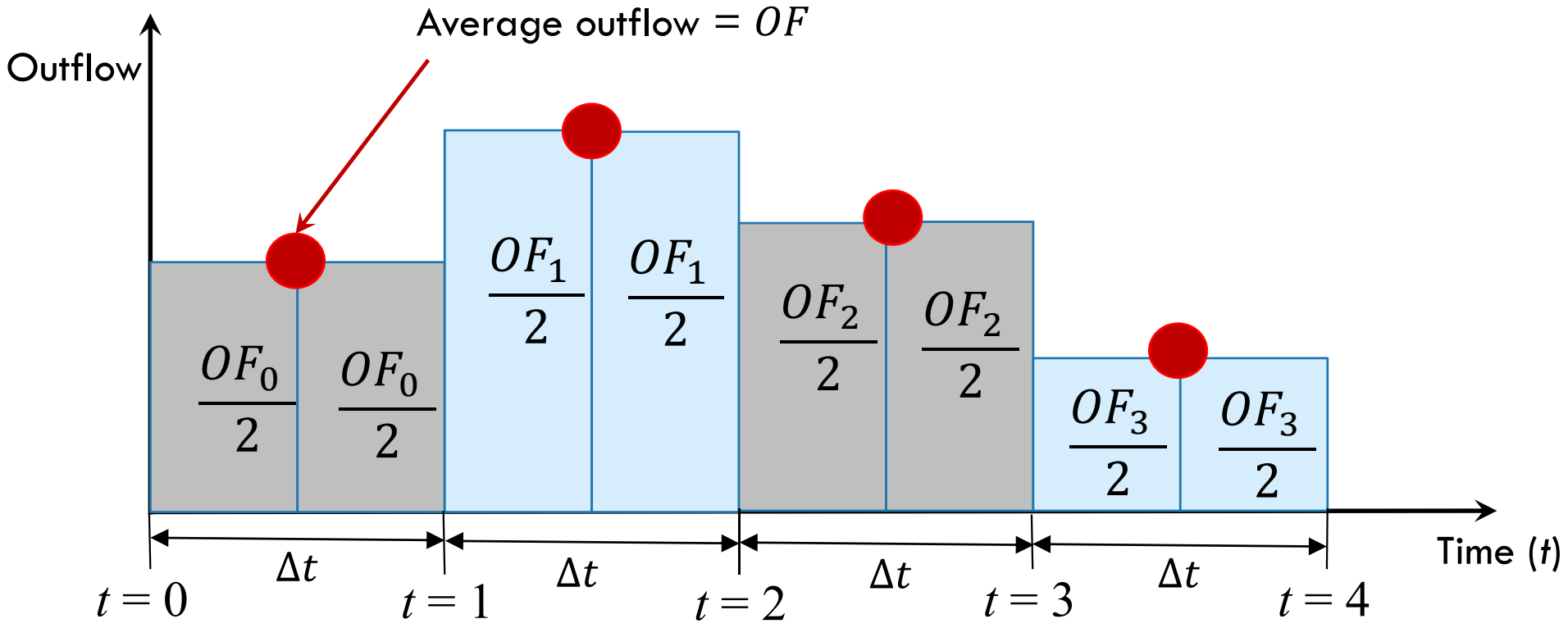
$$S_{3/2} = \underbrace{\text{inloop}_1 + \Delta t\left[\frac{IF_1}{2} + \frac{IF_2}{2}\right]}_{\text{inloop}_2} - \Delta t[TF_1 + SF_1] - \Delta t\left[\frac{TF_2}{2} + \frac{SF_2}{2}\right]$$

STORAGE AT HALF TIME STEP (CONT.)

$$S_{5/2} = \underbrace{\text{inloop}_2 + \Delta t\left[\frac{IF_2}{2} + \frac{IF_3}{2}\right]}_{\text{inloop}_3} - \Delta t[TF_1 + TF_2 + SF_1 + SF_2] - \Delta t\left[\frac{TF_3}{2} + \frac{SF_3}{2}\right]$$

$$S_{t-1/2} = \underbrace{\text{inloop}_{t-1} + \Delta t\left[\frac{IF_{t-1}}{2} + \frac{IF_t}{2}\right]}_{\text{inloop}_t} - \Delta t\left[\sum_{i=1}^{i=t-1}(TFi + SFi)\right] - \Delta t\left[\frac{TF_t}{2} + \frac{SF_t}{2}\right]$$

ELECTRICITY AT FULL TIME STEP



DISCRETIZATION

E : Electricity power in Watts

Substituting $S(t)$ in *E(t)* :

$$E(t) = \mathrm{TF}(t) * [k_1 S(t - 1/2) + k_2]$$

$$E(1) = TF_1\left(k_1 S_{1/2} + k_2\right) = TF_1\left(k_1[\text{inloop}_1] - k_1 \Delta t\left[\frac{TF_1}{2} + \frac{SF_1}{2}\right] + k_2\right)$$

$$E(1) = TF_1(k_1[\text{inloop}_1] + k_2) + TF_1\left(-k_1 \Delta t\left[\frac{TF_1}{2} + \frac{SF_1}{2}\right]\right)$$

$$E(1) = TF_1(k_1[\text{inloop}_1] + k_2) - k_1 \Delta t * TF_1\left(\frac{TF_1}{2} + \frac{SF_1}{2}\right)$$

DISCRETIZATION (CONT.)

Substituting $S(t)$ in *E(t)* :

$$E(2) = \mathrm{TF}(2) * [k_1 S(3/2) + k_2]$$

$$E(2) = TF_2 \left(k_1 [\text{inloop}_2] - k_1 \Delta t (TF_1 + SF_1) - k_1 \Delta t \left[\frac{TF_2}{2} + \frac{SF_2}{2} \right] + k_2 \right)$$

$$E(2) = TF_2 (k_1 [\text{inloop}_2] + k_2) + TF_2 \left(-k_1 \Delta t [TF_1 + SF_1] - k_1 \Delta t \left[\frac{TF_2}{2} + \frac{SF_2}{2} \right] \right)$$

$$E(2) = TF_2 (k_1 [\text{inloop}_2] + k_2) - k_1 \Delta t * TF_2 \left([TF_1 + SF_1] + \left[\frac{TF_2}{2} + \frac{SF_2}{2} \right] \right)$$

DISCRETIZATION (CONT.)

$$E(3) = \mathrm{TF}(3) * [k_1 S(5/2) + k_2]$$

$$E(3) = TF_3(k_1[\text{inloop}_3] + k_2)$$

$$- k_1 \Delta t * TF_3 \left([TF_1 + TF_2 + SF_1 + SF_2] + \left[\frac{TF_3}{2} + \frac{SF_3}{2} \right] \right)$$

In general:

$$E(t) = TF_t(k_1[\text{inloop}_t] + k_2)$$

$$- k_1 \Delta t * TF_t \left(\left[\sum_{i=1}^{i=t-1} (TFi + SFi) \right] + \left[\frac{TF_t}{2} + \frac{SF_t}{2} \right] \right)$$

ELECTRICITY POWER REVENUE

$$R = \Delta t \sum_{i=1}^{t} P_i E_i$$

Where:

R = Revenue

P = Price

$$R_1 = \Delta t\ P_1\ TF_1(k_1[\text{inloop}_1] + k_2) - k_1 \Delta t^2\ P_1 * TF_1(\frac{TF_1}{2} + \frac{SF_1}{2})$$

$$R_2 = \Delta t\ P_2 TF_2(k_1[\text{inloop}_2] + k_2)\ - k_1 \Delta t^2 P_2\ * TF_2\left([TF_1 + SF_1] + \left[\frac{TF_2}{2} + \frac{SF_2}{2}\right]\right)$$

$$R_3 = \Delta t\ P_3 TF_3(k_1[\text{inloop}_3] + k_2) -\ k_1 \Delta t^2 P_3 * TF_3\left([TF_1 + TF_2 + SF_1 + SF_2] + \left[\frac{TF_3}{2} + \frac{SF_3}{2}\right]\right)$$

$$R_t = \Delta t\ P_t TF_t(k_1[\text{inloop}_t] + k_2)\ - k_1 \Delta t^2\ P_t\ * TF_t\left(\left[\sum_{i=1}^{i=t-1}(TFi + SFi)\right] + \left[\frac{TF_t}{2} + \frac{SF_t}{2}\right]\right)$$

$$R = \sum_{i=1}^{t} R_i$$

ELECTRICITY POWER REVENUE (CONT.)

F' * X

$$R = \Delta t \left[\sum_{i=1}^{t} [P_t TF_t (k_1 [\text{inloop}_t] + k_2)] \right]$$

1/2(X' * H * X)

$$- k_1 \Delta t^2 \left[\sum_{i=1}^{i=t} P_t * TF_t \left(\left[\sum_{j=1}^{j=t-1} (TFj + SFj) \right] + \left[\frac{TF_t}{2} + \frac{SF_t}{2} \right] \right) \right]$$

ELECTRICITY POWER REVENUE (CONT.)

$$R = \Delta t \left[\sum_{i=1}^{t} [P_t TF_t (k_1 [\text{inloop}_t] + k_2)]\right] - k_1 \Delta t^2 \left[\sum_{i=1}^{i=t} P_t * TF_t \left(\left[\sum_{j=1}^{j=t-1} (TFj + SFj)\right] + \left[\frac{TF_t}{2} + \frac{SF_t}{2}\right]\right)\right]$$

f' * X

1/2(X' * H * X)

$N = 2$

$$R = \Delta t [P_1 TF_1 (k_1 [\text{inloop}_1] + k_2) + P_2 TF_2 (k_1 [\text{inloop}_2] + k_2)] - k_1 \Delta t^2 \left[P_1 * TF_1 \left(\frac{TF_1 + SF_1}{2}\right) + P_2 * TF_2 \left(TF_1 + SF_1 + \frac{TF_2 + SF_2}{2}\right)\right]$$

F' * X

$N = 2$

$$X' = [TF_1 \;\; TF_2 \;\; SF_1 \;\; SF_2] = [x_1 \;\; x_2 \;\; x_3 \;\; x_4]$$

$$R = \Delta t \sum_{i=1}^{t} [P_t TF_t (k_1 [\text{inloop}_t] + k_2)]$$

F' * X

$$F' * X = \Delta t [P_1 TF_1 (k_1 [\text{inloop}_1] + k_2) + P_2 TF_2 (k_1 [\text{inloop}_2] + k_2)]$$

$$F' * X = \Delta t [P_1 (k_1 [\text{inloop}_1] + k_2) x_1 + P_2 (k_1 [\text{inloop}_2] + k_2) x_2]$$

$$F' * X = \Delta t \begin{bmatrix} P_1 (k_1 [\text{inloop}_1] + k_2) & P_2 (k_1 [\text{inloop}_2] + k_2) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

1/2(X' * H * X)

$$X' = [TF_1 \;\; TF_2 \ldots TF_N \; SF_1 \;\; SF_2 \ldots SF_N] = [\mathrm{x}_1 \;\; \mathrm{x}_2 \ldots \mathrm{xN} \; \mathrm{x}_{N\_+\_1} \ldots \mathrm{x}_{2\mathrm{N}}]$$

$X =$
$[TF_1$
TF_2
...
TF_N
SF_1
SF_2
...
$SF_N]$

Use sym in MATLAB (**simplify**) for multiple number of time steps

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \, \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \, \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \, \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \, \partial x_1} & \frac{\partial^2 f}{\partial x_n \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

X^T in MATLAB is represented as X'!

1/2(X' * H * X)

$N = 2$ (2 time steps)

$$X' = [TF_1 \;\; TF_2 \;\; SF_1 \;\; SF_2] = [x_1 \;\; x_2 \;\; x_3 \;\; x_4]$$

1/2(X' * H * X)

$$-k_1 \Delta t^2 \left[P_1 * TF_1 \left(\frac{TF_1 + SF_1}{2} \right) + P_2 * TF_2 \left(TF_1 + SF_1 + \frac{TF_2 + SF_2}{2} \right) \right]$$

Using sym in MATLAB (**simplify**)

1/2(X' * H * X) = (P1*x1^2)/2 + (P1*x1*x3)/2 +
P2*x1*x2 + P2*x2*x3 + (P2*x2^2)/2 + (P2*x2*x4)/2

1/2(X' * H * X) (CONT.)

N = 3

$X' = [\ x1, x2, x3, x4, x5, x6]$

$$X = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \end{bmatrix}$$

H =

```
[    P1,    P2,    P3, P1/2,     0,     0]
[    P2,    P2,    P3,   P2, P2/2,     0]
[    P3,    P3,    P3,   P3,   P3, P3/2]
[ P1/2,    P2,    P3,    0,     0,     0]
[     0, P2/2,    P3,    0,     0,     0]
[     0,     0, P3/2,    0,     0,     0]
```

1/2(X' * H * X) =

(P1*x1^2)/2 + (P1*x1*x4)/2 +
(P2*x2^2)/2 + P2*x1*x2 + P2*x2*x4 + (P2*x2*x5)/2 +
(P3*x3^2)/2 + P3*x1*x3 + P3*x2*x3 + P3*x3*x4 + P3*x3*x5 + (P3*x3*x6)/2

ASSEMBLING THE HESSIAN

% **File:** dispSymbolicHessianMatrix.m

% N = 4

H =

```
[   P1,    P2,    P3,    P4, P1/2,     0,     0,     0]
[   P2,    P2,    P3,    P4,   P2, P2/2,     0,     0]
[   P3,    P3,    P3,    P4,   P3,   P3, P3/2,     0]
[   P4,    P4,    P4,    P4,   P4,   P4,   P4, P4/2]
[ P1/2,    P2,    P3,    P4,    0,     0,     0,     0]
[    0, P2/2,    P3,    P4,    0,     0,     0,     0]
[    0,     0, P3/2,    P4,    0,     0,     0,     0]
[    0,     0,     0, P4/2,    0,     0,     0,     0]
```

$$\mathbf{H} = \begin{vmatrix} H11 & H12 \\ H21 & H22 \end{vmatrix}$$

H11 = sequential

H21 = H12'

H22 = 0

OPTIMIZING POWER PRODUCTION (CONT.)

Objective function: Maximize the power revenue

Minimize -1/2 * (x' * **H** * x) - f' * x

x = Decision variables

Objective function using symbolic variables in MATLAB:

N = Number of hours of operation

```
X = sym('x',[2*N,1]);   % turbine flow and spill flow
F = sym('F',[N,1]);     % flow into the reservoir
P = sym('P',[N,1]);     % price
s0 = sym('s0');         % initial storage
c = sym({'c1','c2','c3','c4'}.','r');   % constants
```

Objective function using symbolic variables in MATLAB (Cont.)

```
% Total outflow equation
TotFlow = X(1:N)+X(N+1:end); % turbine flow + spill flow

% Storage Equations
S = cell(N,1);
S{1} = s0 + c(1)*(F(1)-TotFlow(1));
for ii = 2:N
S{ii} = S{ii-1} + c(1)*(F(ii)-TotFlow(ii));
end
```

Objective function using symbolic variables in MATLAB (Cont.)

```
% K factor equation
k = c(2)*([s0; S(1:end-1)]+ S)/2+c(3);

% Megawatt-hours equation
MWh = k.*X(1:N)/c(4);

% Revenue equation
R = P.*MWh;
% Total Revenue (Minus sign changes from maximiz. to minimization)
totR = -sum(R);
```

OPTIMIZING POWER PRODUCTION (CONT.)

Constraints:

1) $0 <$ Turbine Flow $< 25{,}000$ CFS
2) $0 <$ Spill Flow
3) Combined Turbine and Spill Flow > 500 CFS
4) Max change in Turbine and Spill Flow < 500 CFS
5) $50{,}000 <$ Reservoir Storage $< 100{,}000$ Acre-Feet
6) The final reservoir level must equal the initial level (90,000 Acre-Feet)

All the constraints are linear so they can be expressed in matrix notation. Our objective, however, is nonlinear.

OPTIMIZING POWER PRODUCTION (CONT.)

Constraints (in MATLAB):

Linear constraints are of the form: $A*x <= b$

Bounds are of the form $LB <= x <= UB$

The hydroelectric dam problem has 4 linear inequality constraints and 2 bound constraints:

Which ones are inequality constraints and bound constraints?

OPTIMIZING POWER PRODUCTION (CONT.)

Constraints (in MATLAB):

```
%% Define Bounds on hourly flows (constraints 1 and 2)
% Bounds are of the form LB <= x <= UB
% lower bound is 0, upper bound is only defined on turbine flow
LB = zeros(2*N,1);     % must have positive flow
UB = [25000*ones(N,1) Inf(N,1)];     % maximum turbine flow of 25,000 CFS, no bound on spill flow
```

OPTIMIZING POWER PRODUCTION (CONT.)

```
%Constraint 3: Minimum Project Flow (2*N decision variables)
% Minimum Project Flow is 500
% turbineFlow(t) + spillFlow(t) >= 500
% -turbineFlow(t) - spillFlow(t) <= -500 (MATLAB convention)
ot = ones(N,1);
b = -500*ot;
A = spdiags([-ot -ot],[0 N],N,N*2);
% A = spdiags(B,d,m,n) creates an m-by-n sparse matrix from the
      columns of B and places them along the diagonals specified by d.
% to see the full matrix use the command full (A)
```

OPTIMIZING POWER PRODUCTION (CONT.)

```
>> full(A)

ans =

    -1     0     0     0    -1     0     0     0
     0    -1     0     0     0    -1     0     0
     0     0    -1     0     0     0    -1     0
     0     0     0    -1     0     0     0    -1
```

```
>> b = -500*ot;
>> full(b)

ans =

  -500
  -500
  -500
  -500
```

OPTIMIZING POWER PRODUCTION (CONT.)

%**Constraint 4:** Change in Project Flow

% Linear constraints are of the form: A*x <= b

% Maximum change in project flow is +/- 500. This constraint is

% represented as:

% -500 <= change <= 500 (i.e. abs(change) <= 500)

% turbineFlow(t)+spillFlow(t)-turbineFlow(t-1)-spillFlow(t-1) <= 500

% -x(turbIndex-1) + x(turbIndex)

% - x(spillIndex-1) + x(spillIndex) <= 500

OPTIMIZING POWER PRODUCTION (CONT.)

```
%Constraint 4:
% constraints for +500
A2 = spdiags([ot ot -ot -ot],[0 N -1 N-1],N,N*2);
b2 = 500*ot;
% remove data for initial condition (There is no N-1 for initial data)
A2(1,:) = [];
b2(1,:) = [];
% now add constraints for the -500 condition
A = [A; A2; -A2]; % N + N-1 + N-1 rows
b = [b; b2; b2]; % N + N-1 + N-1 rows
```

OPTIMIZING POWER PRODUCTION (CONT.)

```
>> full(A2)

ans =

     1     0     0    -1     1     0     0     0
    -1     1     0     0    -1     1     0     0
     0    -1     1     0     0    -1     1     0
     0     0    -1     1     0     0    -1     1
```

```
>> b2 = 500*ot;
>> full(b2)

ans =

   500
   500
   500
   500
```

OPTIMIZING POWER PRODUCTION (CONT.)

```
>> A = [A; A2; -A2];
>> full(A)

ans =

    -1     0     0     0    -1     0     0     0
     0    -1     0     0     0    -1     0     0
     0     0    -1     0     0     0    -1     0
     0     0     0    -1     0     0     0    -1
    -1     1     0     0    -1     1     0     0
     0    -1     1     0     0    -1     1     0
     0     0    -1     1     0     0    -1     1
     1    -1     0     0     1    -1     0     0
     0     1    -1     0     0     1    -1     0
     0     0     1    -1     0     0     1    -1
```

```
>>
>> full(b)

ans =

  -500
  -500
  -500
  -500
   500
   500
   500
   500
   500
   500
```

OPTIMIZING POWER PRODUCTION (CONT.)

```
%Constraint 5: Reservoir Storage Constraints (for each time interval)
% 50000 <= Stor(t) <= 100,000 AF
% Stor(t) = s0+sum(inFlow(1:t))-sum(spillFlow(1:t))-sum(turbineFlow(1:t))
c = stor0 + C2A*cumsum(inFlow); % Convert CFS to AF
b = [b; 100000-c; -50000+c];
s = -C2A*sparse(tril(ones(N)));
s = [s s];
A = [A; s; -s];
```

OPTIMIZING POWER PRODUCTION (CONT.)

```
>> full(inFlow)

ans =

        1070
        1070
        1070
        1070
        1070
        1070
```

```
>> cumsum(inFlow)

ans =

        1070
        2140
        3210
        4280
```

```
>> c = stor0 + C2A*cumsum(inFlow);
>> full(c)

ans =

   1.0e+04 *

    9.0088
    9.0177
    9.0265
    9.0354
```

OPTIMIZING POWER PRODUCTION (CONT.)

```
>> s = -C2A*sparse(tril(ones(N)));
>>
>> full(s)

ans =

   -0.0826         0         0         0
   -0.0826   -0.0826         0         0
   -0.0826   -0.0826   -0.0826         0
   -0.0826   -0.0826   -0.0826   -0.0826
>> s = [s s];
>> full(s)

ans =

   -0.0826         0         0         0   -0.0826         0         0         0
   -0.0826   -0.0826         0         0   -0.0826   -0.0826         0         0
   -0.0826   -0.0826   -0.0826         0   -0.0826   -0.0826   -0.0826         0
   -0.0826   -0.0826   -0.0826   -0.0826   -0.0826   -0.0826   -0.0826   -0.0826
```

```
s = [s s];
A = [A; s; -s];
>>
>> full(A)

ans =

   -1.0000         0         0         0   -1.0000         0         0         0
         0   -1.0000         0         0         0   -1.0000         0         0
         0         0   -1.0000         0         0         0   -1.0000         0
         0         0         0   -1.0000         0         0         0   -1.0000
   -1.0000    1.0000         0         0   -1.0000    1.0000         0         0
         0   -1.0000    1.0000         0         0   -1.0000    1.0000         0
         0         0   -1.0000    1.0000         0         0   -1.0000    1.0000
    1.0000   -1.0000         0         0    1.0000   -1.0000         0         0
         0    1.0000   -1.0000         0         0    1.0000   -1.0000         0
         0         0    1.0000   -1.0000         0         0    1.0000   -1.0000
   -0.0826         0         0         0   -0.0826         0         0         0
   -0.0826   -0.0826         0         0   -0.0826   -0.0826         0         0
   -0.0826   -0.0826   -0.0826         0   -0.0826   -0.0826   -0.0826         0
   -0.0826   -0.0826   -0.0826   -0.0826   -0.0826   -0.0826   -0.0826   -0.0826
    0.0826         0         0         0    0.0826         0         0         0
    0.0826    0.0826         0         0    0.0826    0.0826         0         0
    0.0826    0.0826    0.0826         0    0.0826    0.0826    0.0826         0
    0.0826    0.0826    0.0826    0.0826    0.0826    0.0826    0.0826    0.0826
```

```
>> full(b)

ans =

   1.0e+04 *

   -0.0500
   -0.0500
   -0.0500
   -0.0500
    0.0500
    0.0500
    0.0500
    0.0500
    0.0500
    0.0500
    0.9912
    0.9823
    0.9735
    0.9646
    0.9558
    0.9469
    0.9381
    0.9293
```

OPTIMIZING POWER PRODUCTION (CONT.)

```
%Constraint 6: Equality Constraints (Aeq*x = beq)
% Reservoir storage at the end of the period must be the same as
% initial storage
Aeq = ones(1,2*N);
beq = sum(inFlow);
```

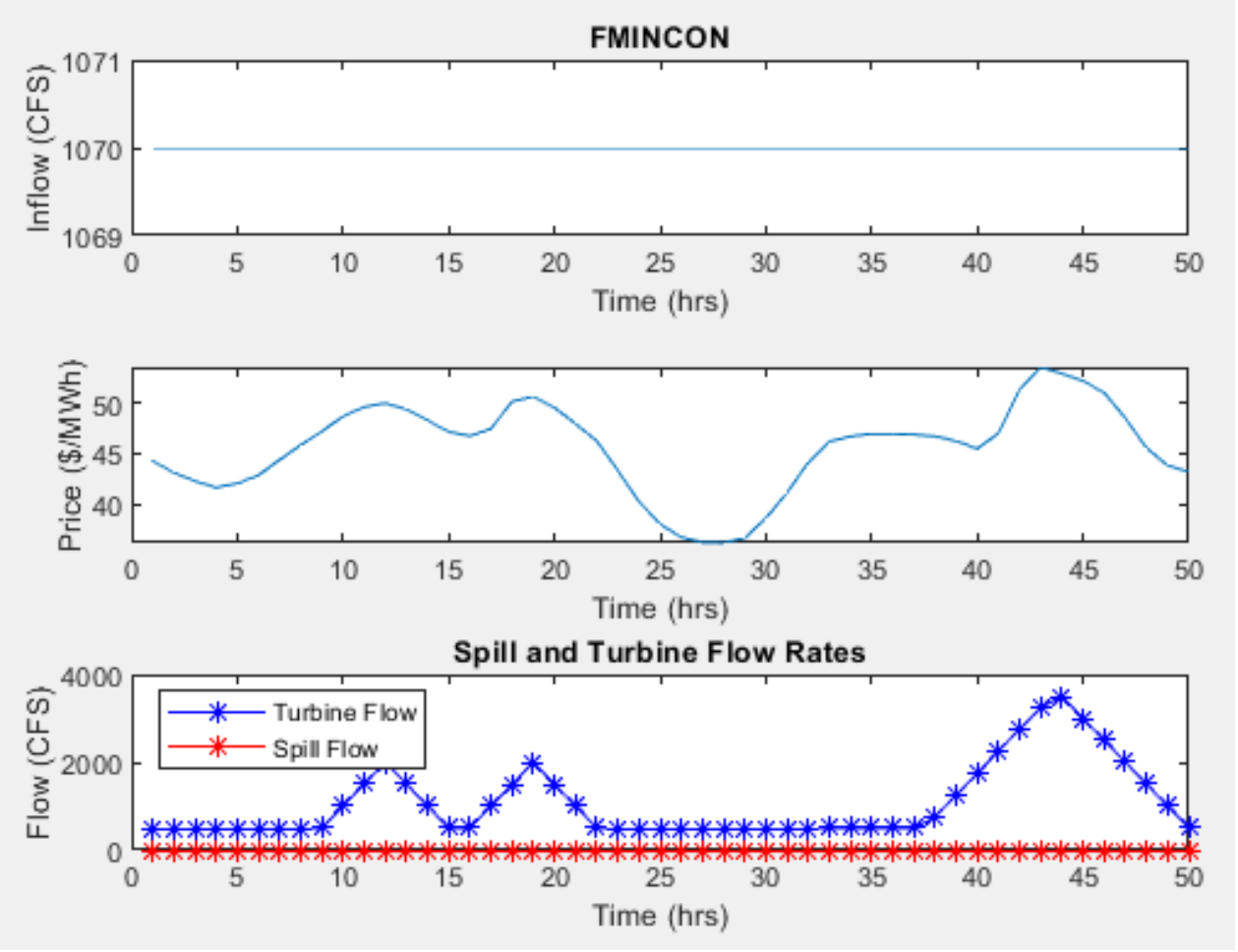
OPTIMIZING POWER PRODUCTION (CONT.)

```
>> Aeq = ones(1,2*N);
beq = sum(inFlow);
>>
>> full(Aeq)

ans =

     1     1     1     1     1     1     1     1

>> full(beq)

ans =

       53500
```

OPTIMIZATION USING FMINCON

```
%% 5 - Optimize with FMINCON
options = optimset('MaxFunEvals',Inf,'MaxIter',5000,...
    'Algorithm','interior-point','Display','iter');

tic
[x1, fval1] =
fmincon(@objFcn,x0,A,b,Aeq,beq,LB,UB,[],options);
toc
% Visualize Results
MethodPlot = 'FMINCON'
plotResults(price, inFlow, x1, N, MethodPlot);
```

OPTIMIZATION USING FMINCON (CONT.)



OPTIMIZ. USING FMINCON WITH SUPPLIED GRADIENT AND HESSIAN

```
Gradient
% Use the GRADIENT function from Symbolic Math to calculate the gradient
gradObj = gradient(totR,X);

% Create a function that when evaluated returns the gradient at a point
matlabFunction(gradObj,'vars',{X},'file','grad');

% The optimization routine expects two return arguments for a user-supplied
% gradient.  The first argument is the value of the objective function and
% the second argument is the gradient.  The DEAL function is used to return
% multiple output arguments through a function handle.
ValAndGrad = @(x) deal(objFcn(x),grad(x));
```

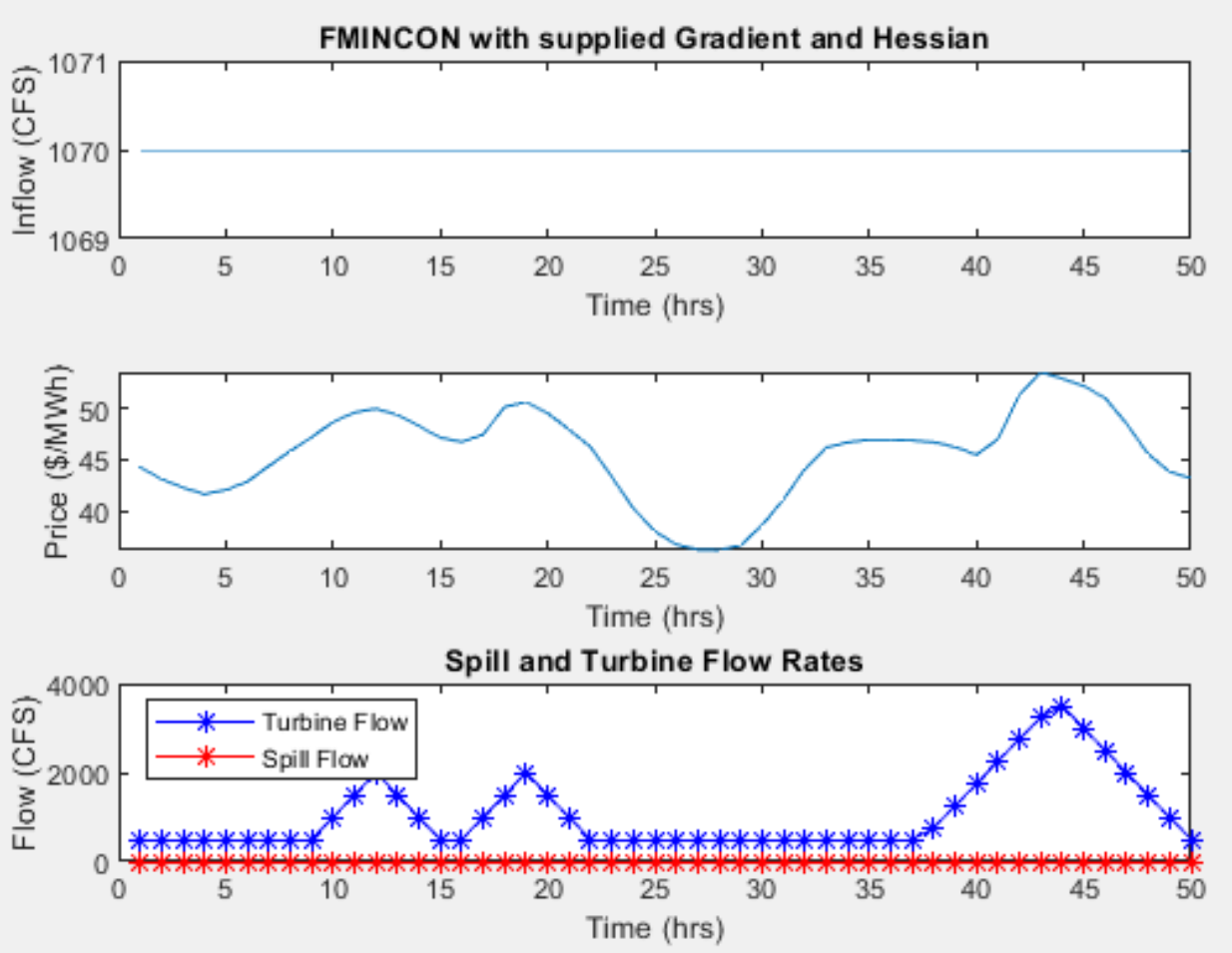
FMINCON WITH SUPPLIED GRADIENT AND HESSIAN (CONT.)

```
%Hessian
% Use the HESSIAN function from Symbolic Math to calculate the Hessian
hessObj = hessian(totR,X);

% Create a function that when evaluated returns the Hessian at a point
matlabFunction(hessObj,'vars',{X},'file','Hess');

% The Hessian function has two input arguments (the current point and the
% Lagrange Multipliers) and returns the value of the Hessian at the current
% point.
hfcn = @(x,lambda) Hess(x);
```

FMINCON WITH SUPPLIED GRADIENT AND HESSIAN (CONT.)

```
% Optimize
options = optimset('Disp','iter','Algorithm','interior-point',...
                   'MaxFunEvals',500,'MaxIter',5000,...
                   'GradObj','on','Hessian','user-supplied',...
                   'HessFcn',hfcn);
tic
[x2,fval2] = fmincon(ValAndGrad,x0,A,b,Aeq,beq,LB,UB,[],options);
toc

% Visualize Results
MethodPlot = 'FMINCON with supplied Gradient and Hessian'
plotResults(price, inFlow, x2, N, MethodPlot);
```

FMINCON WITH SUPPLIED GRADIENT AND HESSIAN (CONT.)



FMINCON WITH SUPPLIED GRADIENT AND HESSIAN (CONT.)

```
% Optimize
options = optimset('Disp','iter','Algorithm','interior-point',...
                   'MaxFunEvals',500,'MaxIter',5000,...
                   'GradObj','on','Hessian','user-supplied',...
                   'HessFcn',hfcn);
tic
[x2,fval2] = fmincon(ValAndGrad,x0,A,b,Aeq,beq,LB,UB,[],options);
toc

% Visualize Results
MethodPlot = 'FMINCON with supplied Gradient and Hessian'
plotResults(price, inFlow, x2, N, MethodPlot);
```

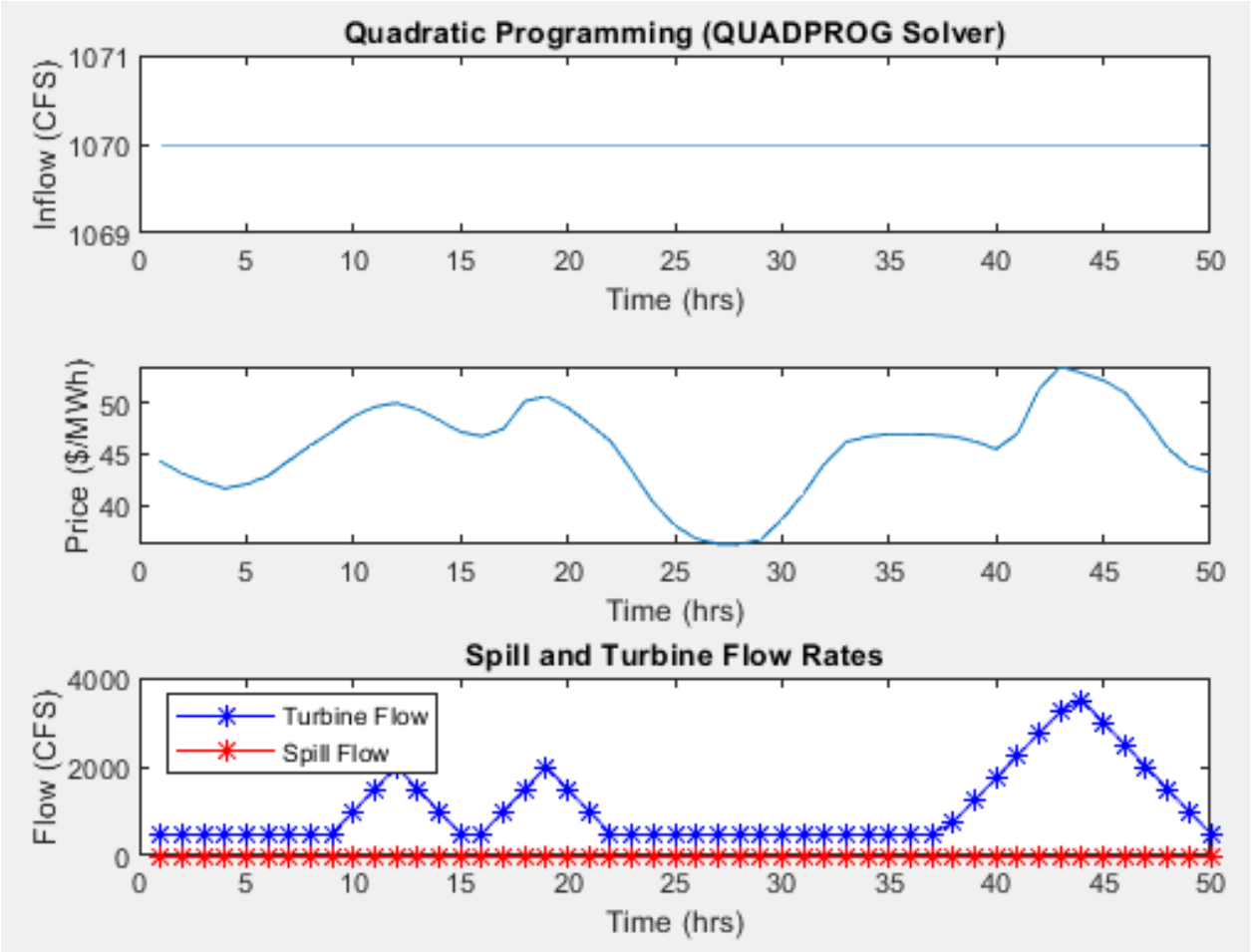
OPTIMIZATION USING THE QUADPROG SOLVER

```
% QUADPROG minimizes functions of the type (1/2) * x' * H * x  +  f' * x
% First perform some substitutions into the revenue equation
Rsub = subs(R,[c;s0;P;F],[C';stor0;price;inFlow]);
% Total revenue (minus sign to flip maximiz. to minimiz.)
Rtot = -sum(Rsub);
%% Calculate and evaluate the linear component
fsym = gradient(Rtot,X);
f = double(subs(fsym,X,zeros(size(X))));
%% Calculate and evaluate the quadratic matrix
H = 0.5.*double(hessian(Rtot,X));
```

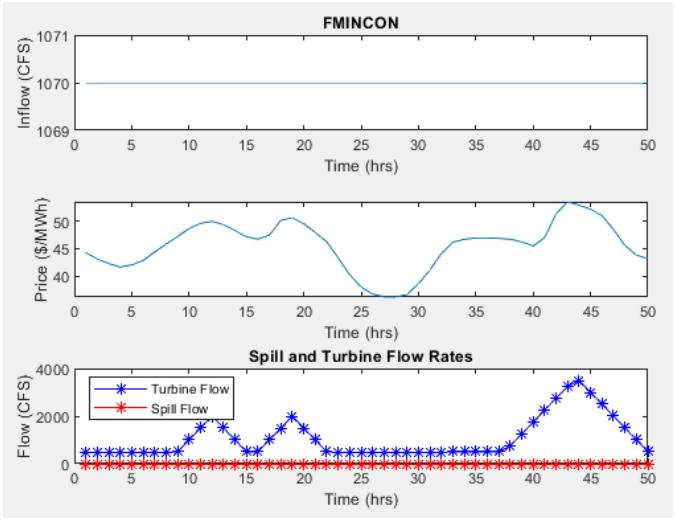
OPTIMIZATION USING THE QUADPROG SOLVER (CONT.)

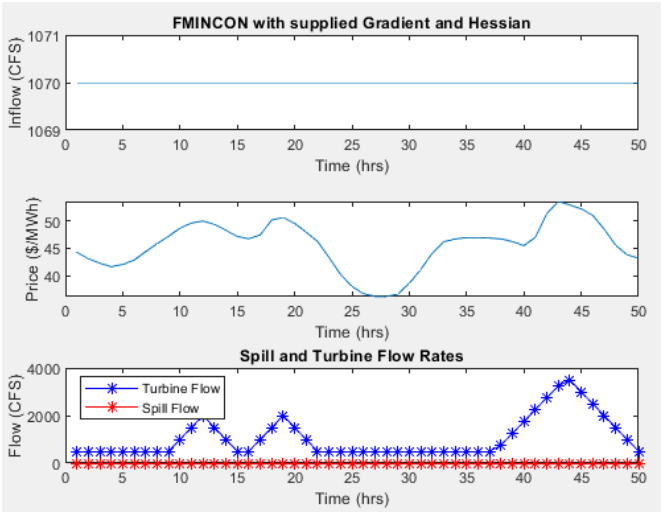
```
% Optimize
qpoptions = optimset('Algorithm','interior-point-convex','Disp','iter');

tic
[x3,fval3] = quadprog(H,f,A,b,Aeq,beq,LB,UB,[],qpoptions);
toc

% Visualize Results
MethodPlot = 'Quadratic Programming (QUADPROG Solver)'
plotResults(price, inFlow, x3, N, MethodPlot);
```

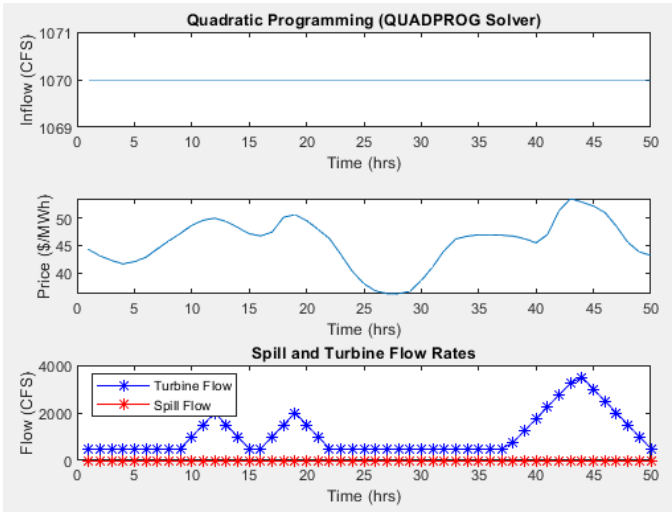
OPTIMIZATION USING THE QUADPROG SOLVER (CONT.)



COMPARISON OF OPTIMIZATION SOLVERS







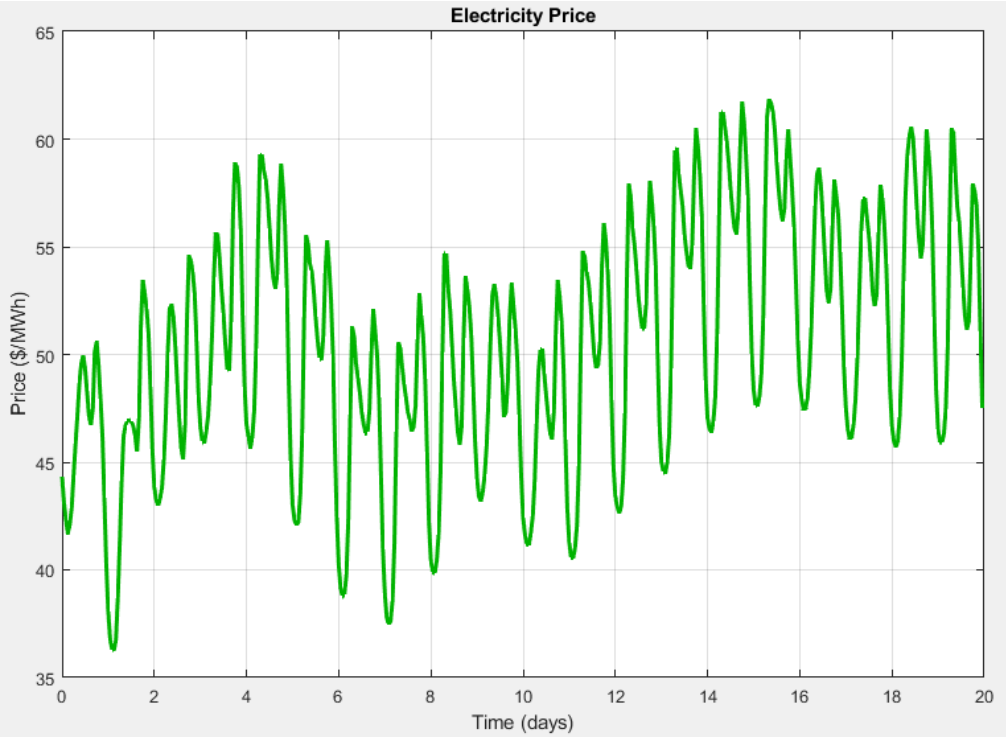
Results from different Optimization solvers (default settings):

Solver	Time(s)
FMINCON	3.525925e+00
FMINCON with supplied Gradient and Hessian	4.209778e-01
Quadratic Programming	2.354854e-01

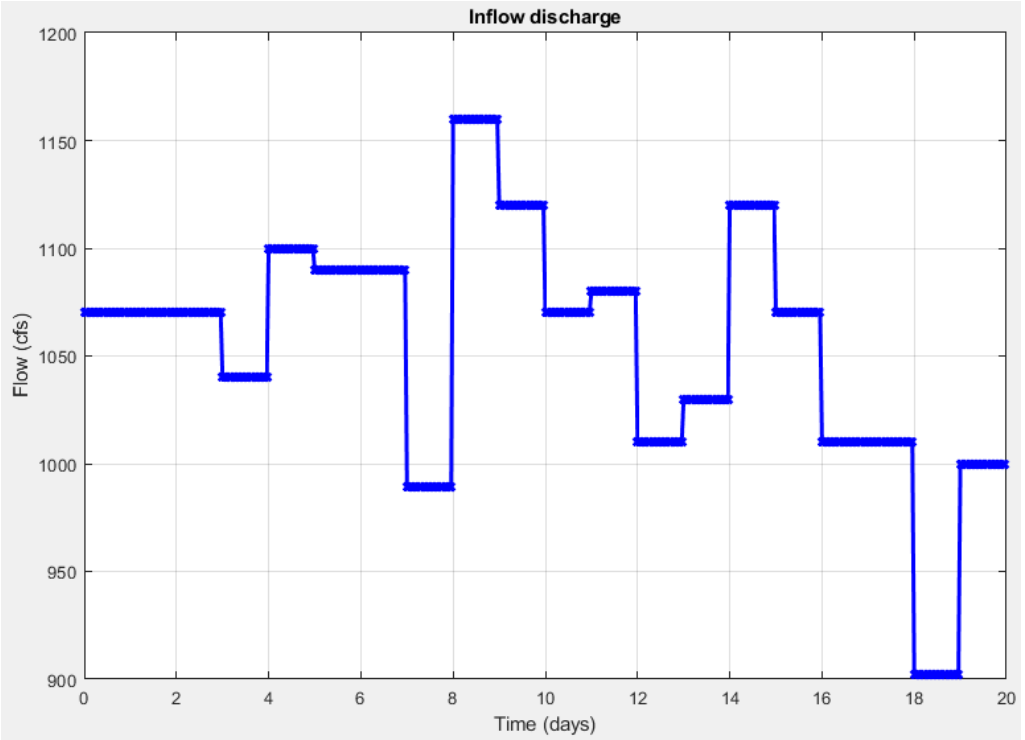
RELATIVELY LARGE SCALE OPTIMIZATION OF HYDROPOWER PRODUCTION

Name of MATLAB File: HydroelectricDamOptimization_largeScale.m

Electricity price



Inflow to reservoir



DEFINE CONSTANTS

```
%Define Constants
stor0 = 90000; % initial vol. of water stored in the reservoir (Acre-Feet)

k1 = 0.00003; % K-factor coefficient
k2 = 9; % K-factor offset

MW2kW = 1000; % Convert from MW to kW
C2A = 1.98347/24; % Convert from CFS to AF/HR
```

OBJECTIVE FUNCTION FOR QUADRATIC PROGRAMMING

```
% Formulate Objective Function for Quadratic Programming
% QUADPROG minimizes problems of the form: 1/2 * (x' * H * x)  +  f' * x
% Because we would like to maximize our profits, we need to formulate our
% maximization problem as a minimization problem:
%
% Minimize   -1/2 * (x' * H * x)  -  f' * x
%
% For this problem, x is split up into two halves:
%   1) Hourly turbine flow
%   2) Hourly spill flow
```

HESSIAN

% **File:** dispSymbolicHessianMatrix.m

% N = 4 (P = Price)

H11 = sequential

H21 = H12'

H22 = 0

```
>> dispSymbolicHessianMatrix

(c1*c2/c4) *

[   P1,    P2,    P3,    P4, P1/2,     0,     0,     0]
[   P2,    P2,    P3,    P4,   P2, P2/2,     0,     0]
[   P3,    P3,    P3,    P4,   P3,    P3, P3/2,     0]
[   P4,    P4,    P4,    P4,   P4,    P4,    P4, P4/2]
[ P1/2,    P2,    P3,    P4,    0,     0,     0,     0]
[    0, P2/2,    P3,    P4,    0,     0,     0,     0]
[    0,     0, P3/2,    P4,    0,     0,     0,     0]
[    0,     0,     0, P4/2,    0,     0,     0,     0]
```

"*f*" vector

```
% Formation of f Vector
% The f vector for this problem is such that each value is dependent on the
% previous values in the vector.  This can be implemented with a FOR loop.
% At each iteration, the variable "inloop" is updated from its previous
% value to the new value.

% Initialize
f = zeros(1,2*N);

% Initialize inloop, it will be incremented each iteration of the for loop
inloop = k1*(stor0 + C2A*inFlow(1)/2);
```

"*f*" vector (Cont.)

```
% Fill in f(1) first
f(1) = price(1)*(inloop + k2);

% Use a loop to fill in the rest of f
for ii = 2:N
    inloop = inloop + k1*C2A*(inFlow(ii-1)/2 + inFlow(ii)/2);
    f(ii) = price(ii)*(inloop + k2);
end
% Convert f from kW to MW and flip sign for maximization
f = -f/MW2kW;
% Create a sparse vector (f is 50% sparse)
f = sparse(f);
```

CONSTRAINTS AND BOUNDS

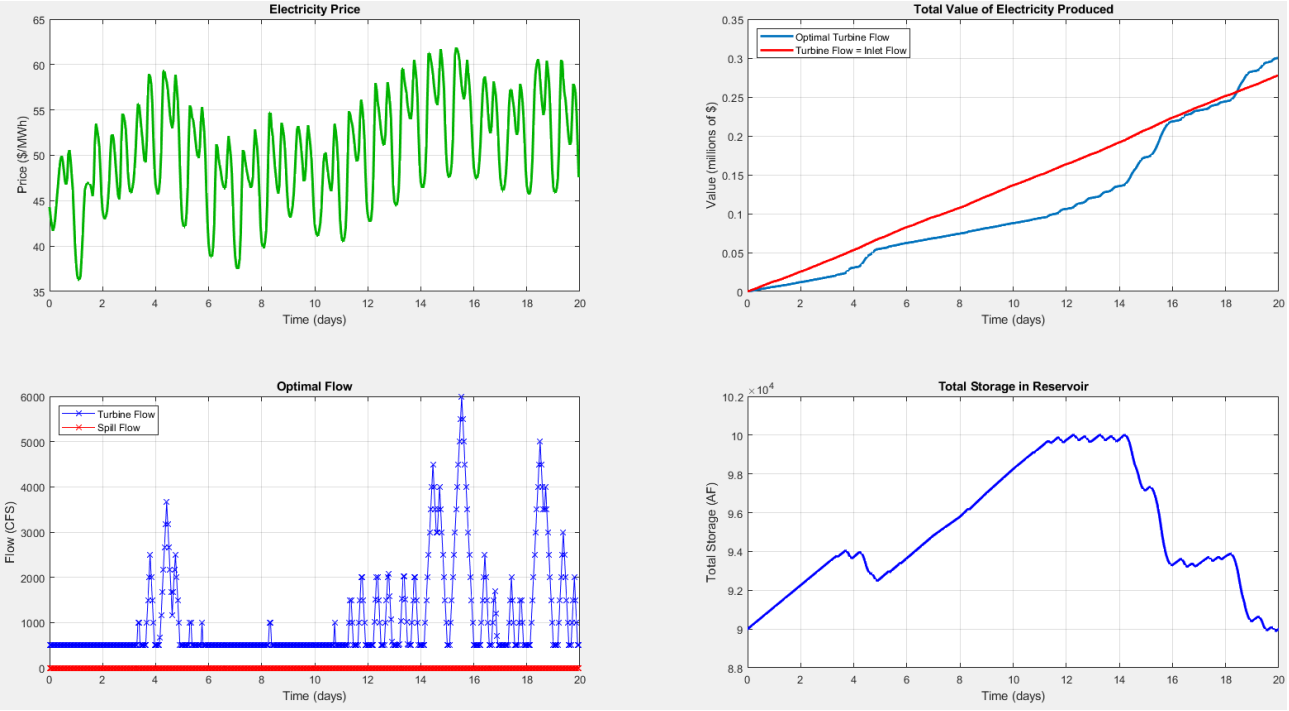
```
%% Define Linear Inequality Constraints and Bounds
DefineConstraints; %% Show in MATLAB how to change this
```

SOLVER AND DISPLAY OF RESULTS

```
% Minimize the function: Choose the Algorithm to be interior-point-convex
qpopts = optimset('Algorithm','interior-point-convex','Display','iter');
 % Perform the optimization
[x,fval] = quadprog(H,f',A,b,Aeq,beq,LB,UB,[],qpopts);

%Display Results
% Extract Turbine flow and Spill flow from x
turbFlow = x(1:N); spillFlow = x(N+1:end);

% Plotting Helper Function
createPlots(turbFlow,spillFlow,inFlow,price,stor0,C2A,N);
```

HANDS-ON DEMONSTRATION

Change constraints for turbine and spill flows, reservoir storage limits, initial reservoir storage (**DefineConstraints.m**)



COMPARISON OF CPU TIMES FOR VARIOUS OPTIMIZATION METHODS