

Non-Linear Programming

General Approach: Finding and comparing peaks and valleys in the objective function within a feasible region defined by the constraint set.

Example Applications

- Profit maximization with price elasticity
- Transportation problems with volume discounts on shipping costs
- Portfolio selection with risky investments (also a multi-objective programming problem)
- Farm management considering agricultural production functions
- Hydropower scheduling
- Truss design

Abbreviated Outline of Methods

Unconstrained Nonlinear Optimization

- Analytical methods
Calculus
- Searching without derivatives
Fibonacci search method
Grid search method
Genetic algorithms
- Searching with first derivatives
One-dimensional (line) search procedure
Gradient search (steepest descent) method (one of many “hill-climbing” methods)
Quasi-Newton methods
Conjugate gradient method
- Searching with second derivatives
Newton’s method
Trust region methods

Constrained Nonlinear Optimization (Nominally, these are globally optimal for min. of convex functions over convex feasible regions.)

- Analytical methods
Lagrange multiplier method
- Exterior penalty function methods
- Interior (barrier) function methods
- Gradient projection methods
- Generalized reduced gradient methods
- Successive linear programming methods
- Successive quadratic programming methods
- Genetic algorithms (constraints represented in “fitness” function)

Algorithms for Special Cases of NLP (Nominally, these are globally optimal.)

- Separable programming (linear/mixed integer approximation)
- Quadratic programming (for min. convex functions)
- Dynamic programming

Non-Linear Search Procedures

Like the simplex method for linear programming, *search procedures* for nonlinear programming problems find a sequence of *trial solutions* that lead to an optimal solution. In each iteration, you begin with the *current* trial solution and proceed systematically to a *new improved* trial solution. Typically, you stop when either one of the following conditions are met:

- (1) Your trial solutions have *converged* to a given value (i.e., successive solutions show little or no improvement).
- (2) Your solution satisfies certain *optimality criteria*, such as the Karush-Kuhn-Tucker conditions.

Fibonacci Search: An efficient method of finding the minimum of a single-variable function $f(x)$ which strictly increases on either side of an optimum value (i.e., is *unimodal*), considering a number of discrete values of the decision variable. In a sequence of n Fibonacci numbers, F_n , the next number in the sequence is the sum of the previous two numbers, e.g., 1, 1, 2, 3, 5, 8, 13, 21, 34.... Where there are $(F_n - 1)$ ordered discrete values of the decision variable X with values x_j , the Fibonacci search can find the minimum of the unimodal function $f(X)$ in $(n-1)$ iterations.

For a minimization search over the interval a_1 to b_1 :

Initialization: Select a final solution tolerance, $t > 0$. Set index $i = 1$
 $L_1 = a_1 + (F_{n-2}/F_n)(b_1 - a_1)$ and $U_1 = a_1 + (F_{n-1}/F_n)(b_1 - a_1)$

Step 1: If $f(L_i) < f(U_i)$, then:

The optimal value is below U_i .
Set $a_{i+1} = a_i$ and $b_{i+1} = U_i$ and
Set $L_{i+1} = a_{i+1} + (F_{n-i-2}/F_{n-i})(b_{i+1} - a_{i+1})$ and $U_{i+1} = L_i$

Step 2: If $f(L_i) \geq f(U_i)$, then:

The optimal value is above L_i .
Set $a_{i+1} = L_i$ and $b_{i+1} = b_i$ and
Set $L_{i+1} = U_i$ and $U_{i+1} = a_{i+1} + (F_{n-i-1}/F_{n-i})(b_{i+1} - a_{i+1})$

Step 3: Set $i = i + 1$

Step 4: If $i < n-1$, go to Step 1

Step 5: If $f(L_i) > f(U_i)$, then:

Final solution lies above L_i , so final solution interval is $[L_i, b_i]$.
ELSE,
Final solution lies below L_i , so final solution interval is $[a_i, U_i]$.

This method is often used in determining the step size (or number of steps, given a specified step size) in the gradient search method, described below. Wagner (1975, p. 455) has a nice example. Bazaraa, et al (1993) have excellent write-ups on this and related search methods.

Grid Search Method: A grid search evaluates points evenly spaced over a suitable grid in the decision space. While this method is easy to program, it is too time-consuming for problems with more than a few decision variables. However, it can find an approximate solution or a good starting point for a more intensive search algorithms.

One-dimensional (line) search procedure: Starting with a trial solution to a minimization problem, one moves in the direction of the gradient (first derivative). (If the first derivative (slope) is positive, move to the left; if it is negative, move to the right.) Below the procedure is summarized using the *mid-point rule*:

Define: x' = current trial solution
 \underline{x} = current lower bound on x^*
 \bar{x} = current upper bound on x^*
 ϵ = error tolerance for x^*

Initialization: Select ϵ . Find initial values for \underline{x} and \bar{x} by inspection (or by finding any values of x at which the derivatives are positive and then negative). Select an initial trial solution:

$$x' = \frac{\underline{x} + \bar{x}}{2}$$

Iteration:

1. Evaluate $\left. \frac{df(x)}{dx} \right|_{x'}$
2. If $\frac{df(x')}{dx} \geq 0$, set $\bar{x} = x'$.
3. If $\frac{df(x')}{dx} \leq 0$, set $\underline{x} = x'$.
4. Select a new $x' = \frac{\underline{x} + \bar{x}}{2}$.

Stopping rule: If $\bar{x} - \underline{x} \leq 2\epsilon$, Stop. Otherwise, perform another iteration.

This is a simple procedure in one dimension. For multiple decision variables, it can be applied to each variable in turn. This is also known as the *univariate gradient search*.

Gradient search (steepest ascent/descent) method: This is another iterative method (based on first-order derivatives) which works easily for multiple dimensions. As in the one-dimensional search procedure, the each iteration moves the trial solution in the direction of the gradient (i.e., downhill for a minimization problem). Additionally, one needs to decide *how far* to move downhill (the step size). At the optimal solution, $\nabla f(\mathbf{x}^*) = 0$.

Consider: $\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]$
 t = step size (a single variable used in the 1-D search)

The gradient search procedure for a *minimization problem* is summarized as:

Initialization: Select ϵ and an initial trial solution \mathbf{x}' . Go to the stopping rule.

Iteration: 1. Express $f(\mathbf{x}' + t\nabla f(\mathbf{x}'))$ as a function of t by setting

$$x_j = x'_j + t \left(\frac{\partial f}{\partial x_j} \right)_{\mathbf{x}=\mathbf{x}'} \quad \text{for } j = 1, \dots, n$$

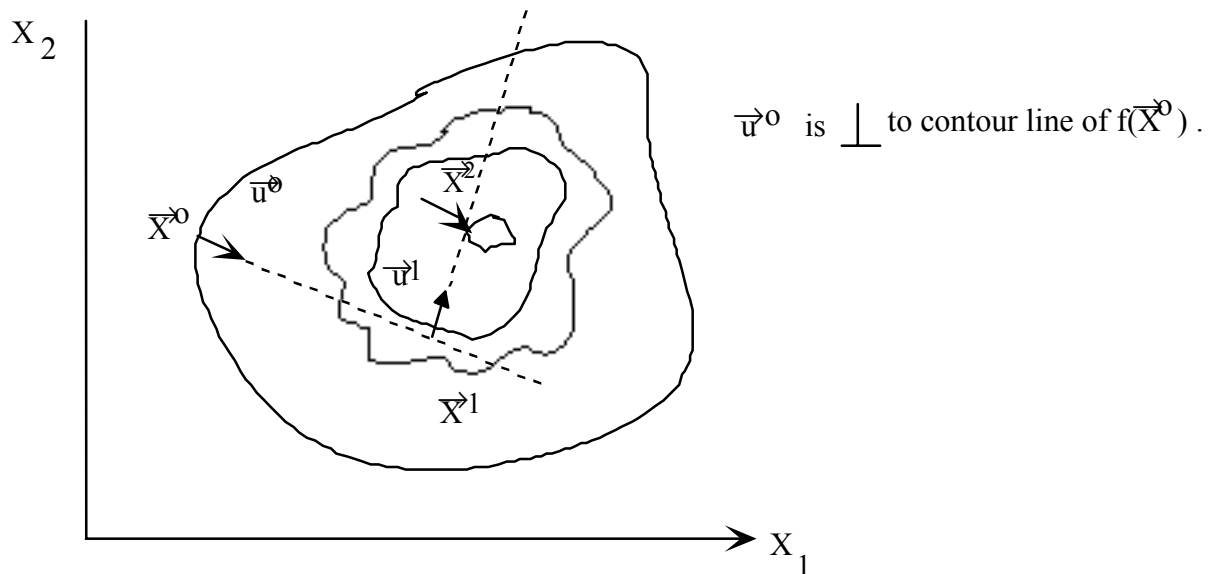
and then substituting these expressions into $f(\mathbf{x})$.

2. Use the one-dimensional line search procedure (or calculus or Fibonacci search) to find $t = t^*$ that minimizes $f(\mathbf{x}' + t\nabla f(\mathbf{x}'))$ over $t \geq 0$.
3. Reset $\mathbf{x}' = \mathbf{x}' + t^* \nabla f(\mathbf{x}')$ and go to the stopping rule.

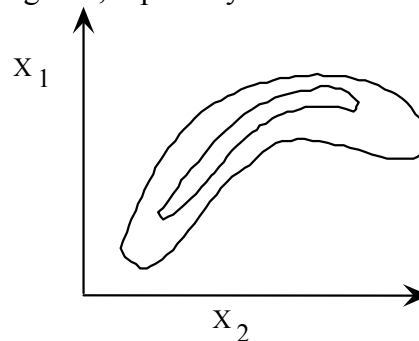
Stopping rule: Evaluate $\nabla f(\mathbf{x}')$. Check if

$$\left| \frac{\partial f}{\partial x_j} \right| \leq \varepsilon \quad \text{for all } j = 1, \dots, n.$$

If so, stop with the current \mathbf{x}' as the approximate optimal solution. Otherwise, perform another iteration.



As do most of the search methods discussed here, the gradient search only finds a local optimum. It also can have very slow convergence, especially for banana-shaped functions. *Why?*



Newton's Method: An iterative search method that uses both first and second derivatives. Like many other search methods, the direction of the iteration is found using the first derivative (slope). The method then cleverly uses the second derivative at a point to predict how far to step to where the first derivative equals zero.

Derivation of Method:

Begin your search at a point X^0 , with $f(X^0)$ = value of the objective function.

The Taylor series of $f(X)$ at X^0 is:

$$f(X^0 + \Delta X) = f(X^0) + \frac{df(X^0)}{dX} \Delta + \frac{1}{2} \frac{d^2 f(X^0)}{dX^2} \Delta^2 + \dots$$

Let $X^1 = X^0 + \Delta X$ be your next estimate of the optimal solution X^* .

Since $\frac{df}{dX} = 0$ at any max or min, express $\frac{df}{dX}$ as a Taylor series, set = 0. The first-order Taylor series

of $\frac{df}{dX} = 0$ at X^1 is:

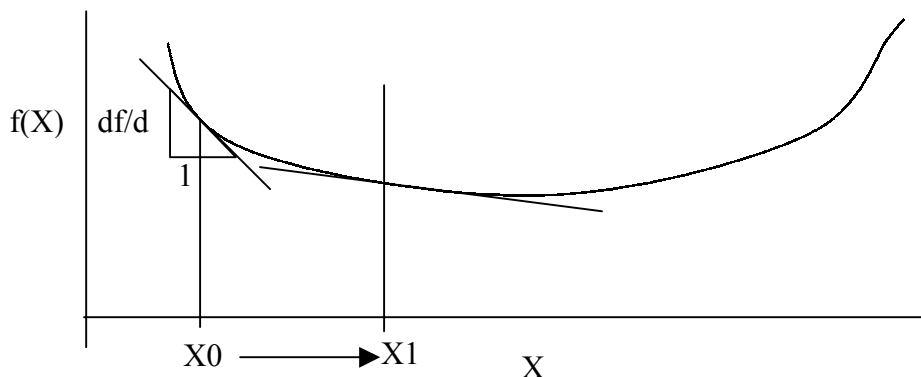
$$\frac{df(X^1)}{dX} = 0 = \frac{df(X^0)}{dX} + \frac{d^2 f(X^0)}{dX^2} \Delta X.$$

Manipulating, this becomes: $\Delta X = -\frac{\frac{df(X^0)}{dX}}{\frac{d^2 f(X^0)}{dX^2}}$, or

$$X^1 = X^0 + \Delta X = X^0 - \frac{\frac{df(X^0)}{dX}}{\frac{d^2 f(X^0)}{dX^2}}$$

Repeat the use of this equation until $\left| \frac{\partial f}{\partial x_j} \right| \leq \epsilon$ for all $j = 1, \dots, n$. Graphically,

What if there is more than one local optimum or more than one decision variable?
The selection of the initial/starting solution is important.



Newton's Method for n-decision Variables

This just applies the ideas of Newton's method to n-decision dimensions.

1st-order condition for minimum or maximum of $f(\mathbf{X})$:

$$\frac{\partial f(\mathbf{X})}{\partial X_i} = 0 \quad \text{for } i = 1, \dots, n.$$

Let \mathbf{X}^0 = initial trial solution, and consider the (multi-dimensional) Taylor series expansions:

$$\frac{\partial f(\mathbf{X}^0 + \Delta\mathbf{X})}{\partial X_i} = 0 \approx \frac{\partial f(\mathbf{X}^0)}{\partial X_i} + \left[\frac{\partial^2 f(\mathbf{X}^0)}{\partial X_i \partial X_j} \right]^T \Delta\mathbf{X}, \quad i = 1, \dots, n.$$

With n decision variables (\mathbf{X}_j), there are n simultaneous equations like the one above:

$$0 = \left[\frac{\partial f(\mathbf{X}^0)}{\partial X_i} \right] + \begin{bmatrix} \frac{\partial^2 f(\mathbf{X}^0)}{\partial X_1^2} & \frac{\partial^2 f(\mathbf{X}^0)}{\partial X_1 \partial X_2} & \dots & \frac{\partial^2 f(\mathbf{X}^0)}{\partial X_1 \partial X_n} \\ \frac{\partial^2 f(\mathbf{X}^0)}{\partial X_2 \partial X_1} & \frac{\partial^2 f(\mathbf{X}^0)}{\partial X_2^2} & & \vdots \\ \vdots & & \frac{\partial^2 f(\mathbf{X}^0)}{\partial X_i^2} & \\ \frac{\partial^2 f(\mathbf{X}^0)}{\partial X_n \partial X_1} & \dots & & \frac{\partial^2 f(\mathbf{X}^0)}{\partial X_n^2} \end{bmatrix}^T \Delta\mathbf{X},$$

In the second term, the large matrix is $\mathbf{H}(\mathbf{X}^0)$, our old friend the Hessian matrix. We can solve for $\Delta\mathbf{X}$ using the inverse of the Hessian matrix:

$$\Delta\mathbf{X} = -\mathbf{H}(\mathbf{X}^0)^{-1} \cdot \left[\frac{\partial f(\mathbf{X}^0)}{\partial X_i} \right]$$

Putting this into an iterative search scheme for n decision variables,

$$\mathbf{X}^1 = \mathbf{X}^0 - \mathbf{H}(\mathbf{X}^0)^{-1} \cdot \left[\frac{\partial f(\mathbf{X}^0)}{\partial X_i} \right]$$

Continue iterations until:

$$\left| \frac{\partial f(\mathbf{X}^0)}{\partial X_i} \right| \leq \varepsilon$$

where ε is a convergence tolerance.

Sequential Linear Programming: An efficient technique for solving convex programming problems with nearly linear objective and constraint functions. As the name suggests, each of the approximating problems will be an LP problem that can be solved relatively efficiently. Also, successive problems will differ only by one constraint, so the dual simplex method can be used to solve the sequence of problems very efficiently.

General procedure:

1. Start with an initial trial solution, \mathbf{x}_i ($i = 0$).
2. Use a first-order Taylor series approximation to linearize the objective and constraint functions about the point \mathbf{x}_i :

$$f(\mathbf{x}) \approx f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^T (\mathbf{x} - \mathbf{x}_i)$$

$$g(\mathbf{x}) \approx g(\mathbf{x}_i) + \nabla g(\mathbf{x}_i)^T (\mathbf{x} - \mathbf{x}_i)$$

3. Formulate the approximating linear programming problem as

$$\text{Minimize } f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^T (\mathbf{x} - \mathbf{x}_i)$$

Subject to

$$g_j(\mathbf{x}_i) + \nabla g_j(\mathbf{x}_i)^T (\mathbf{x} - \mathbf{x}_i) \leq 0 \quad \text{for } j = 1, \dots, m.$$

4. Solve the approximating LP to find a new trial solution \mathbf{x}_{i+1} .
5. Evaluate the original constraints at \mathbf{x}_{i+1} ; that is, find

$$g_j(\mathbf{x}_{i+1}), \quad j = 1, \dots, m.$$

Example Sequential LP problem:

$$\text{Max } 3X_1 + 5X_2 + 10 \cos(X_3)$$

$$\text{S.T. } X_1 + X_2 + X_3 \leq 20$$

-) Search over X_3 (enumeration, Fibonacci, Newton, or Gradient Search)
-) LP for each value of X_3 tried.

Quasi-Global Optimum Search Methods

Relies on finding and comparing several local optima by starting local searches in different areas of the feasible region.

Once a local optimum has been found, how can a new starting point be found that makes finding a different local optimum most likely?

Approaches:

- 1) Eye-ball it, manually choosing starting points from different parts of the feasible region.
- 2) Randomly select new starting points.
- 3) Select a new starting point which maximizes the distance from all previously-examined points.

Step 1: Start a local optimum search starting from $\vec{X}^{0,1}$ and ending with $\vec{X}^{F_{1,1}}$; $\vec{X}^{F_{1,1}} = \vec{X}^{*,1}$.

Remember the best solution found \vec{X}^* and its objective function value f^* .

Step 2: If k previous local searches have been undertaken over n decision variables; solve:

$\text{Max } Z = \sum_{j=1}^k \sum_{i=1}^n \sum_{l=1}^{F_j} X_i - X_i^{l,k} $ $\text{S.T. } \vec{g}(\vec{X}) \leq \vec{b}$

Step 3: Use the solution to this linear program as a new starting point for a local search.

Step 4: Go to step 2.

The above linear program finds a point a maximum distance from all points previously examined.
If $Z^* \leq \epsilon$, STOP; Enough area has been searched.

-) Computationally intensive
-) Often only local optima are found.
-) Harder to program; fewer nice solution packages.

Example:

Modeling to Generate Alternatives using HSJ.

Brill, E.D., S.Y. Chang, and L.D. Hopkins (1982), "Modeling to generate alternatives: The HSJ approach," *Management Science*, Vol. 28, No. 3, pp. 221-225.

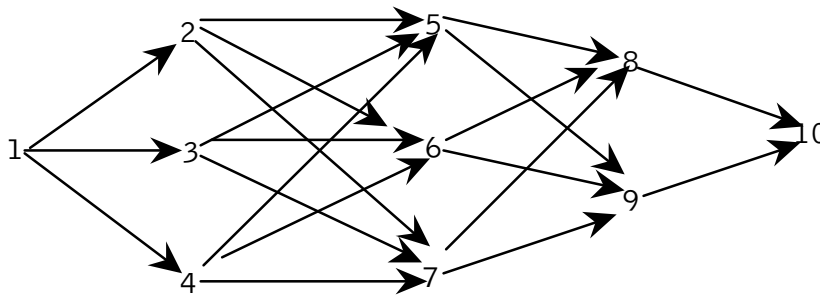
Dynamic Programming (DP)

Dynamic programming is a method of solving a particular class of math. programming problems. The method is essentially an efficient form of enumeration, somewhat like the branch-and-bound method of integer programming.

Illustrate method by way of a problem:

Example: The stage-coach problem; (starts p.320 of Hillier and Lieberman text)

A salesman wants to get from location 1 to location 10 with a minimum of risk, as represented by life insurance premiums on each intermediate journey. His map:



Note that the total trip is divided into 4 stages:

1 to 2,3, or 4; 2,3, or 4 to 5,6, or 7; 5,6, or 7 to 8 or 9; and 8 or 9 to 10.

The life insurance cost for each possible trip segment. C_{ij} of going from i to j .

Stage 1	Stage 2	Stage 3	Stage 4
<u>2</u> <u>3</u> <u>4</u>	<u>5</u> <u>6</u> <u>7</u>	8 9	<u>10</u>
1 →: 2 4 3	2 →: 7 4 6 3 →: 3 2 4 4 →: 4 1 5	5 →: 1 4 6 →: 6 3 7 →: 3 3	8 →: 3 9 →: 4

How to select a route with minimum total life insurance cost?

1) By enumeration: Compare total cost of each possible combination of routes.

- a) 1,2,7,8,10
- b) 1,2,7,9,10
- c) 1,2,6,8,10
- d) 1,2,6,9,10
- e) 1,2,5,8,10
- f) 1,2,5,9,10
- g) 1,3,7,8,10

etc. (18 possible combinations of routes) Lots of combinations work.

2) Use a short-sighted method. Choose the cheapest route for the first stage, $1 \rightarrow 2$ (\$2); then the cheapest route from 2 to the 2nd stage, $2 \rightarrow 6$ (\$4); then again for the third stage, $6 \rightarrow 9$ (\$3); and the fourth stage, $9 \rightarrow 10$ (\$4). The total cost is then \$13. $1 \rightarrow 2 \rightarrow 6 \rightarrow 9 \rightarrow 10$: \$13

But, $1 \rightarrow 4 \rightarrow 6 \rightarrow 9 \rightarrow 10$ costs only $3 + 1 + 3 + 4 = \$11$

So this method will not guarantee the optimal solution.

3) Dynamic Programming

Note that the routing decision is divided into 4 stages. At each stage only one next-step of several possible steps can be chosen.

For Stage 4: The last decision:

Make a table:

n = 4	S	$f_4^*(S) = c_{ij}$	X_4^*
	8	3	10
	9	4	10

X_4^* is the decision variable of which location to go to next.

X_4^* is always 10, the end.

S is the state in which the stage 4 decision is made (the entering state or condition). Go from 8 or 9.

$f_4^*(S)$ is the cost of going from the present possible state to the final state.

So far this isn't very interesting.

Let's now work backwards, to the next-to-last stage, stage 3.

n = 3	S	$f_3(S_3, X_3) = c_3(X_3) + f_4^*(X_3)$			$f_3^*(S)$	X_3^*
		$X_3: 8$	9			
	5	4	8	4	8	
	6	9	7	7	9	
	7	6	7	6	8	

-) Going from 5 to 10, it is best to pass through 8.

-) Going from 6 to 10, it is best to pass through 9.

-) Going from 7 to 10, it is best to pass through 8.

$f_i()$ = recursive objective function, "cost-to-go" function, or accumulated objective function. It has two parts, 1) the direct costs of the decision in the current stage and state and 2) the best implied cost for all the later states, $f_{i+1}^*(S_{i+1})$.

Backwards again to Stage 2.

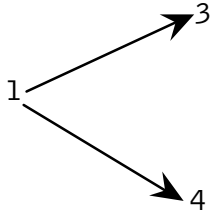
n = 2	S	$f_2(S_2, X_2) = c_2(X_2) + f_3^*(S_3 = X_2)$			$f_2^*(S)$	X_2^*
		$X_2: 5$	6	7		
	2	11	11	12	11	5 or 6
	3	7	9	10	7	5
	4	8	8	11	8	5 or 6

Same interpretation . Give examples.

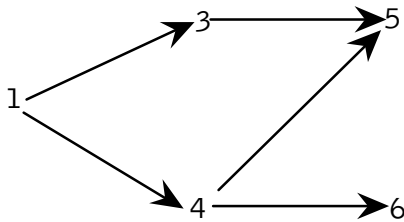
Finally, back again to Stage 1.

n = 1	S	$f_1(S_1, X_1) = c_1(X_1) + f_2^*(S_2 = X_1)$			$f_1^*(S)$	X_1^*
		$X_1: 2$	3	4		
	1	13	11	11	11	3 or 4

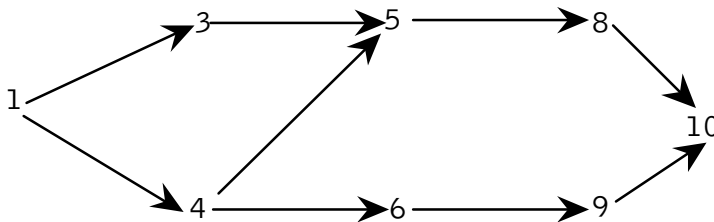
The best decision in Stage 1 is to go either to loc. 3 or 4. This is the overall optimum.



Looking at Stage 2 results, note that if coming from loc. 3, going to loc. 5 is preferred and if coming from loc. 4 go either to loc. 5 or 6.



Then for Stages 3 and 4:



There are 3 equally-optimal paths:

- 1,3,5,8,10
 - 1,4,5,8,10, and
 - 1,4,6,9,10.
- All have total costs of \$11.

Why did this work?
 Elimination of inferior subsets of decisions.
 Go through stages to show why.

DP Theory

In solving this or any other DP problem we must formulate problems into:

Stages: A decision must be made at each stage.

States: When making a decision at a stage, there are a number of states (or conditions) we can be in when making the decision. (State variables must be discrete.)

Decision Variable and options: At each stage we have a limited number of choices. (Decision variables must be discrete.)

Objective function: Objective function

Recursive Accumulated Objective Function: for usual "backwards recursion":

$$f_n(S, X_n) = c_n(S, X_n) + f_{n+1}^*(S_{n+1} = h(S, X_n))$$

$S_{n+1} = h(S, X_n)$ is the "state transition" equation

DP works by accumulating the optimal objective function as it moves through stages, eliminating bad decisions. (the "recursive relationship").

At the last stage:

-) One optimal decision is specified, and
-) The optimal objective function value is found.

The rest of the optimal decisions are found by tracing from states to decisions for other stages.

Backward DP

Example: World Health Council Example (p. 339 in Hillier and Lieberman text).

The WHC has 5 medical teams it can allocate among 3 countries. It would like to allocate these teams to maximize the additional person-years of life (A.P.Y.L.) for all countries combined. (A.P.Y.L. = increased life expectancy • Population) [NOTE: Another way is to increase population while keeping life expectancy the same.]

Table 1 gives the return (in 1,000's of person-yrs) for each country.

[X _i]	No. of Teams	1,000's person-years return for each country			[P _i (X _i)]
		1	2	3	
	0	0	0	0	←
	1	45	20	50	
	2	70	45	70	
	3	90	75	80	
	4	105	110	100	
	5	120	150	130	

Formulated as a math program:

$$\text{Max } z = \sum_{i=1}^3 P_i(X_i)$$

$$\text{S.T. } \sum_{i=1}^3 X_i = 5$$

$$X_i = \text{integer}, \forall i$$

[Could solve by integer programming.]

Stages: Let each country be a stage. The decision in Stage 3 will be how many teams go to Country 3?

States: The incoming state of each will be the number of teams remaining to be allocated.

Decision: How many teams should go to the country in a given stage if only 5 teams are remaining to be allocated?

Objective function: Max
$$z = \sum_{i=1}^3 P_i(X_i)$$

Accumulated objective function:

$$f_n(S, X_n) = P_n(X_n) + f_{n+1}^*(S - X_n)$$

the "recursive relationship"

Go over

Again, we'll solve it backwards, starting with Stage/County 3.

Stage 3

n = 3	S	$f_n^*(S)$	X_2^*
	0	0	0
	1	50	1
	2	70	2
	3	80	3
	4	100	4
	5	130	5

Essentially, allocate all that remains at the last Stage to Country 3.

Stage 2

n = 2	$f_2(S, X_2) = P_2(X_2) + f_3^*(S - X_2)$						$f_2^*(S)$	X_2^*
	S / X_2 :	0	1	2	3	4		
0	0	--	--	--	--	--	0	0
1	50	20	--	--	--	--	50	0
2	70	70	45	--	--	--	70	0 or 1
3	80	90	95	75	--	--	95	2
4	100	100	115	125	110	--	125	3
5	130	120	125	145	160	150	160	4

$$\begin{aligned} f_3(4,2) &= P_2(2) + f_3^*(4-2=2) \\ &= 45 + 70 \\ &= 115 \end{aligned}$$

Stage 1: At Stage 1, no allocations have been made, so $S = 5$.

$n = 1$	$f_1(S, X_1) = P_1(X_1) + f_2^*(S - X_1)$						$f_1^*(S)$	X_1^*
S:	$X_1: 0$	1	2	3	4	5		
5	160	170	165	160	155	120	170	1

\downarrow
 $= P_1(1) + f_2^*(5-1=4)$
 $= 45 + 125$
 $= 170$

The Solution

The maximum number of additional person-years is 170,000.

Trace the optimal decisions forward through the stages:

Stage i	X_i^*	S	$S - X_i^*$
1	1	5	$4 = 5 - 1$
2	3	4	$\leftarrow 1 = 4 - 3$
3	1	1	$\leftarrow 0 = 1 - 1$

Compare Calculations Needed for Enumeration Vs. DP

Typical comparison is by number of cost calculations needed to solve the problem.

For DP: No. of Cost Calculations =

$$\sum_{\text{Stage } n = 1}^{n_{\max}} (\text{number of states in stage } n) \cdot (\text{number of decisions examined})$$

if S_{\max} and d_{\max} are same for all stages.

$$\text{No. of cost cal.} = n_{\max} \cdot S_{\max} \cdot d_{\max}$$

For enumeration:

No. of cost cal. =

$$\prod_{n=1}^{n_{\max}} d_{n, \max}$$

If d_{\max} is same for all stages:

$$\text{No. cost cal.} = d_{\max}^{n_{\max}}$$

For WHC problem:

For DP: No. cost calc. = $6(1) + (6)(6) + 1(6) = 48$
 (including infeasible sol'ns)
 $= 6 + 21 + 6 = 33$
 (excluding infeasible sol'ns)

By Enumeration: No. of cost calc. - $6 \cdot 6 \cdot 6 = 6^3 = 216$
 (including infeasible sol'ns)

DP requires 22% the calculation effort for this problem.

WHC Example Revisited.

Forward Accumulated Objective function:

$$f_n(S_{n+1}, X_n) = P_n(X_n) + f_{n-1}^*(S_n = S_{n+1} + X_n)$$

Concern of outgoing rather than incoming state.

Stage 1: n = 1

Incoming state: $S_1 = 5$

$$f_1(X_1) = P_1(X_1) = P_1(S_1 - S_2)$$

$$X_1 = S_1 - S_2$$

Outgoing State S_2	f_1^*	X_1^*
0	120	5
1	105	4
2	90	3
3	70	2
4	45	1
5	0	0

Stage 2

n = 2

$$f_2(S_3, X_2) = P_2(X_2) + f_1^*(S_3 + X_2 = S_2)$$

Outgoing State S_3	X_2 :						$f_2^*(S)$	X_2^*
	0	1	2	3	4	5		
0	$105 + 20$ 120 = 125 135 145 155 150						155	4
1	$70 + 45$ 105 110 = 115 120 110 --						120	3
2	$45 + 45$ 90 90 = 90 75 -- --						90	0,1 or 2
3	70 65 45 -- -- --						70	0
4	45 20 -- -- -- --						45	0
5	0 -- -- -- -- --						0	0

Stage 3

n = 3

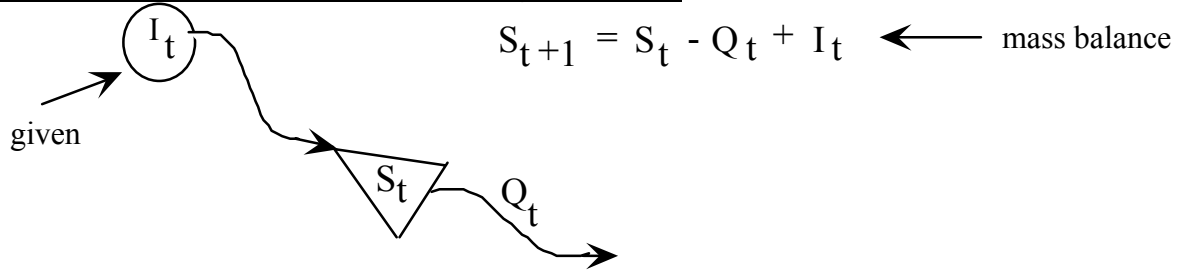
$$f_3(S_4, X_3) = P_3(X_3) + f_2^*(S_4 + X_3)$$

Outgoing State S_4	X_3 :						$f_3^*(S_4)$	X_3^*
	0	1	2	3	4	5		
0	$0 + 155$ $120 + 50$ $90 + 70$ $70 + 80$ $45 + 100$ $0 + 130$ = 155 = 170 = 160 = 150 = 145 = 130						170	1

1,3,1 it works

More DP Applications

Reservoir Releases Over Time from a Single Reservoir



The DP:

Stages: time-periods, t

States: Volume of water in reservoir at time t , S_t

Decisions: How much water to release at time t , Q_t

Objective function: Max economic benefits = $\sum_{t=1}^{t_n} b_t(S_t, Q_t)$

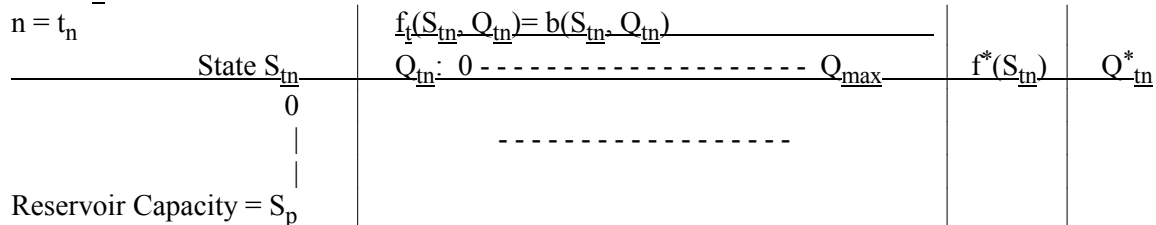
Cumulative objective function: Backward moving:

$$f_t(S_t, Q_t) = b(S_t, Q_t) + f_{t+1}^*(S_{t+1} = S_t - Q_t + I_t)$$

Start with $t = t_n$ and move stages backwards to $t = 1$.

If $Q_t > S_t + I_t$, then $b(S_t, Q_t) = -\infty$; infeasible.

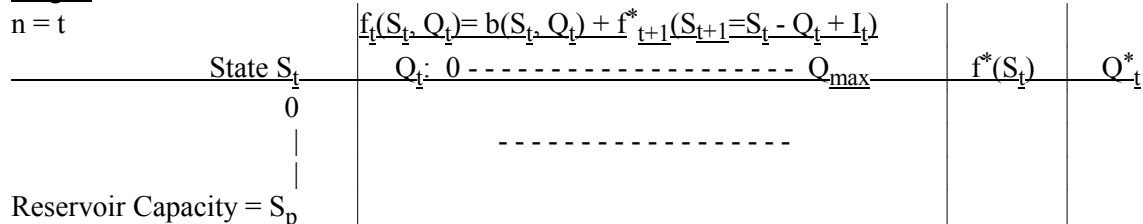
Stage t_n



Must discretize both -) Release values (m values)

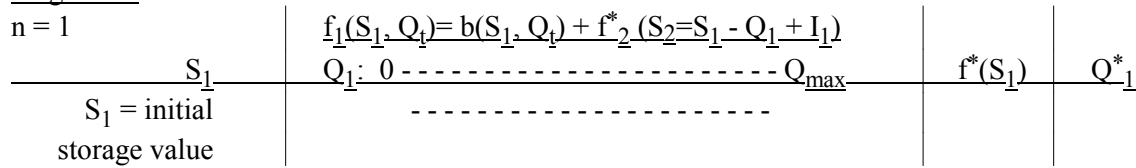
_____ -) Storage values (p values)

Stage t



Stage $t = 1$

$n = 1$



There is only one reservoir state in the present, S_1 .

What would a forwards formulation be?

$$f_t(S_t, Q_t) = b(S_t, Q_t) + f_{t-1}^*(S_{t-1} = S_t + Q_t - I_t), \text{ where } S_t \text{ is the storage/state leaving stage } t.$$

Computational effort to solve this reservoir problem:

Let t_n = number of stages

m = number of decision alternatives per stage

p = number of discretizations of the state variable

Number of cost calculations by DP approx. $t_n * p * m$

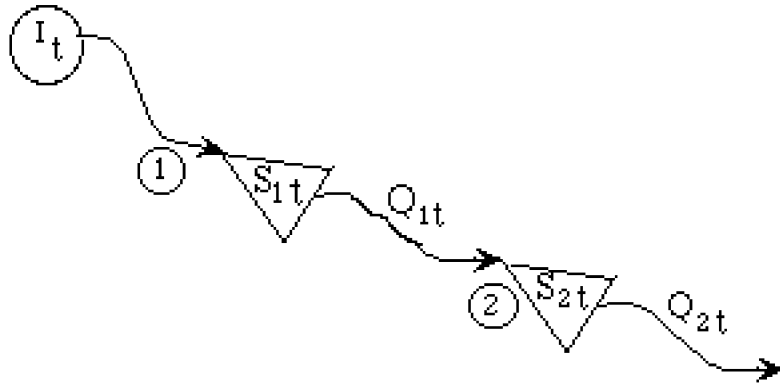
Number of cost calculations by enumeration: approx. m^{t_n}

Let's compare the computational effort for $t_n = 7$ days, and $p = m$.

Discretizations of m and p	Calculations By DP	Calculations by enumeration
10	$7 \cdot 10 \cdot 10 = 700$	10,000,000
100	$70,000 = 7 \cdot 10^4$	10^{14}
1,000	$7,000,000 = 7 \cdot 10^6$	10^{21}

It's possible to solve a much larger reservoir problem by DP than would otherwise be possible.

Operating 2 Reservoirs in Series:



Stages: Time-periods t

State 1: Storage in Reservoir 1: discretized values

State 2: Storage in Reservoir 2: discretized values

Decision 1: Release from Reservoir 1: discretized values

Decision 2: Release from Reservoir 2: discretized values

Cumulative objective function: Backward moving:

$$f_t(S_{1t}, S_{2t}, Q_{1t}, Q_{2t}) = b_t(S_{1t}, S_{2t}, Q_{1t}, Q_{2t}) + f_{t+1}^*(S_{1t} + I_t - Q_{1t}, S_{2t} + Q_{1t} - Q_{2t})$$

Note: Conservation of mass equations for each reservoir determine the transition between states at each stage. $S_{1t+1} = S_{1t} + I_t - Q_{1t}$ and $S_{2t+1} = S_{2t} + Q_{1t} - Q_{2t}$.

To set-up for each stage:

Stage t		$f_t(S_{1t}, S_{2t}, Q_{1t}, Q_{2t})$			$f^*_t(S_{1t}, S_{2t})$	Q^*_{1t}	Q^*_{2t}
States		$Q_1: 0$	1	-----			
S_1	S_2	$Q_2: 0$	1 2 3 --- m	0 1 2 3 --- m	-----		
0	0						
	1						
	2						
	p						
1	0						
	1						
	2						
	p						
2							
3							
p							

Computational Effort:

$\text{No. of cost calculations by DP} = t_n \cdot p_1 \cdot p_2 \cdot m_1 \cdot m_2$

For 2 reservoirs over 7 days, let $p_1 = p_2 = m_1 = m_2$

Discretizations of m	Calculations By DP	Calculations by enumeration
10	70,000	10^{14}
100	$700,000,000 = 7 \cdot 10^8$	10^{28}
1,000	$7 \cdot 10^{12}$	10^{42}

Calculations by enumeration = $m^{2 \cdot 7}$

General Comments:

-) Multiple State Variables
-) Multiple Decision Variables
-) "Curse of Dimensionality"

The Final Days

-) Multi-Objective Optimization
-) Viewing operations research in terms of Alternative Generation & Comparison
 - to suggest solutions
 - to get people talking
 - need to test and refine decisions suggested by optimization methods
-) Review again the steps of O.R. application.
-) Uses of operations research in practice

Pre-Opt. & Post-Opt.

Go back and put methods in context of Operations Research method.

Pre-Opt.: (use notes from early in class)

- 1) Formulating the Problem as an O-R-problem
 -) What is the problem?
 -) Objective function definition(s)
 -) Decision variables
 -) Constraints
 -) Simplifications identified
- 2) Solve Mathematical Problem
- 3) Interpret Results (perhaps modify, re-solve, and re-interpret)

Overview/Review of Solution Methods (Make a huge table)

- 1) Formulation limitations
- 2) Computational limitations
- 3) Advantages
- 4) Disadvantages
- 5) How does this method work?

for:

Trial & Error

Enumeration

Calculus

Lagrange Multipliers

LP

Integer-LP

Non-Linear Programming

DP

GA

Discussion: why select one method over another.