Multidimensional Optimization

- Life get <u>much</u> more complex for higher (N>1) dimensional optimization.
- In general, we start from a given point, pick a <u>search direction</u>, and do a 1-D search in that direction.
- <u>Method of steepest descent</u>: If we want to get to a minimum, it makes sense to go downhill.

  Search direction = – grad(F)

Multidimensional Optimization

- Thus we solve the 1-D problem:

$$\min_{\alpha} F\{\mathbf{x}^{(k)} - \alpha\, \mathbf{grad}[F(\mathbf{x}^{(k)})]\}$$

to get:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_{opt}\, \mathbf{grad}\, F(\mathbf{x}^{(k)})$$

where α_{opt} is the desired minimum.

- This method tends to be rather slow. The gradient often does not point towards the minimum!

Multidimensional Optimization

- Let's work an example:

  Let $F(\mathbf{x}) = x_1^2 + 10x_2^2 + 100x_3^2$
- This has a global minimum of $\mathbf{x}^* = \mathbf{0}$

  $(x_1^* = x_2^* = x_3^* = 0)$
- Now grad $F = (2x_1, 20x_2, 200x_3)^T$
- Thus at each iteration we want to solve:

  $\min_\alpha F(\mathbf{x} - \alpha\, \mathbf{grad}\ F) =$

  $\min_\alpha \{(x_1 - 2\alpha x_1)^2 + 10(x_2 - 20\alpha x_2)^2 + 100(x_3 - 200\alpha x_3)^2\}$

Multidimensional Optimization

- We can actually get a linear equation for α for this particular problem.

  $\alpha = (x_1^2 + 10^2 x_2^2 + 10^4 x_3^2)/(2(x_1^2 + 10^3 x_2^2 + 10^6 x_3^2))$

- After <u>180</u> iterations we get:

  $x^{(0)} = (1,1,1)$ ← starting point

  $x^{(180)} = (1.07 \times 10^{-4}, 0, 2.01 \times 10^{-6})$

- Why? The problem has different curvature in different directions.
- You can think of the algorithm as wandering back and forth across a narrow river valley and slowly rolling down to the sea.

Multidimensional Newton's Method

- We can do better (sometimes) with a <u>multi-dimensional Newton's method</u>.
- Keep an extra term in the Taylor series:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_0) + \nabla^{\mathrm{T}} f(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^{\mathrm{T}} \boldsymbol{H}(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0),$$

- The matrix ***H*** is

  the <u>Hessian matrix</u>:

$$\boldsymbol{H}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ & \vdots & & \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Multidimensional Newton's Method

- The quadratic approximation is differentiated with respect to x and set equal to zero:

$$\boldsymbol{H}(\boldsymbol{x}_0)\Delta\boldsymbol{x} = -\nabla f(\boldsymbol{x}_0),$$

where $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_0$.

- If we truncate after this term, we get an equation for the next guess at the critical point!

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \left[\boldsymbol{H}\left(\boldsymbol{x}^{(k)}\right)\right]^{-1}\nabla f\left(\boldsymbol{x}^{(k)}\right).$$

- These are Newton's equations for this problem!

Multidimensional Newton's Method

- The method will converge quadratically near the critical point, but it can fail to converge.
- Suppose we want to find a minimum. We want an algorithm that does the following:
  1. Computes $F(\mathbf{x}^{(k)})$, $\mathbf{grad}\ F(\mathbf{x}^{(k)})$
  2. Computes some descent direction $\mathbf{p}$ such that:

     $F(\mathbf{x}^{(k)} + \varepsilon\mathbf{p}) < F(\mathbf{x}^{(k)})$

     for small ε.

     We can do this via steepest descent:

     $\mathbf{p} = -\mathbf{grad}\ F(\mathbf{x}^{(k)})$

Multidimensional Optimization

or via Newton's method:

$$\mathbf{p} = -\left[\boldsymbol{H}\left(\boldsymbol{x}^{(k)}\right)\right]^{-1}\nabla f\left(\boldsymbol{x}^{(k)}\right).$$

Note that Newton's method does not <u>necessarily</u> point to a minimum! (max, saddle)

3. Line search along vector **p**:

   Find some α such that

   $F(\mathbf{x}^{(k)} + \alpha\mathbf{p})$ is minimized.

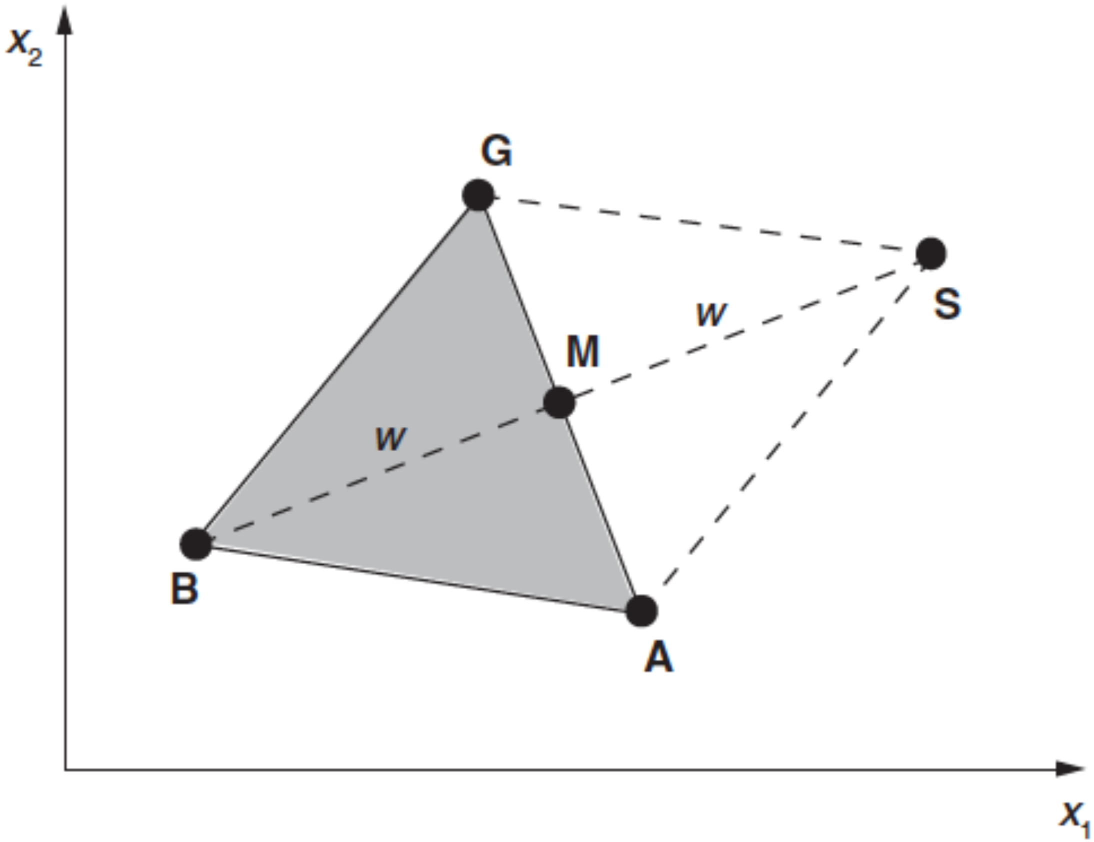
   This is a 1-D optimization problem!

Then return to step (1).

Simplex Method

- A completely different algorithm is the simplex method.
- This algorithm uses a search based on triangles in 2-space and multi-dimensional pyramids in n-space.
- We look at the 2-D problem…

Construction of a new triangle **GSA** (simplex). The original triangle is shaded in gray. The line **BM** is extended beyond **M** by a distance equal to *w*



- We pick 3 points in the form of an equilateral triangle.
- We discard the largest point (when looking for a minimum) and pick a new point which is its mirror image.

- We then wander through space until the minimum is reached!

Simplex Method

- Technique works well if all elements of **grad** F are comparable.
- Tip: rescale variables in an ill-conditioned problem so that this is realized.
- Method is modified when minimum is approached (size of triangle is reduced).
- Nelder-Mead implementation also changes the triangle shape → makes triangle longer in the direction of the minimum.
- Matlab function fminsearch uses this approach.

Q1: Calculating the <u>condition number</u> of a matrix is an easy way to determine whether a matrix is singular (and thus, cannot be inverted). Which result would indicate a singular matrix in Matlab?

cond(J) =

A. 0.000
B. 1.500
C. 107.3
D. 4.280e+16

```
>> J=[1 4;2 8]

J =

     1     4
     2     8

>> cond(J)

ans =

   4.2799e+16

>> inv(J)
Warning: Matrix is singular to
working precision.

ans =

   Inf   Inf
   Inf   Inf
```

Q2: \ Backslash or left matrix divide

The expression:

$$A\backslash B =$$

is equivalent to what operation in Matlab?

A. inv(A)*B
B. A*B
C. A*inv(B)
D. A.*B
E. A/B

```
>> help \
 \   Backslash or left matrix divide.
    A\B is the matrix division of A into
B, which is roughly the same as INV(A)*B
, except it is computed in a different
way.
    If A is an N-by-N matrix and B is a
column vector with N components, or a
matrix with several such columns, then X
= A\B is the solution to the equation
A*X = B.
    A warning message is printed if A is
badly scaled or nearly singular.
```

Q3: function gen_newtonsmethod2(func, x, tolx, tolfx)

```
% func   : function that evaluates the system of nonlinear equations and
         % the Jacobian matrix
% x      : initial guess values of the independent variables
% tolx   : tolerance for error in the solution
% tolfx  : tolerance for error in function value

% Other variables
maxloops = 20;

[fx, J] = feval(func,x);
fprintf(‘   i    x1(i+1)    x2(i+1)    f1(x(i))    f2(x(i))   \n’);
% Iterative solution scheme
for i = 1:maxloops
    dx = J \ (-fx);
    x = x + dx;
    [fx, J] = feval(func,x);
    fprintf(‘%2d  %7.6f  %7.6f  %7.6f  %7.6f  \n’, . . .
      i, x(1), x(2), fx(1), fx(2));
    ?????????????????????????????????
         % not use of element-wise AND operator
         break % Jump out of the for loop
    end
end
```

A. fx = feval(func,dx)

B. if (abs(dx) <=tolx & abs(fx) <tolfx)

C. if(abs(tolx) <=dx & abs(tolfx) <fx)

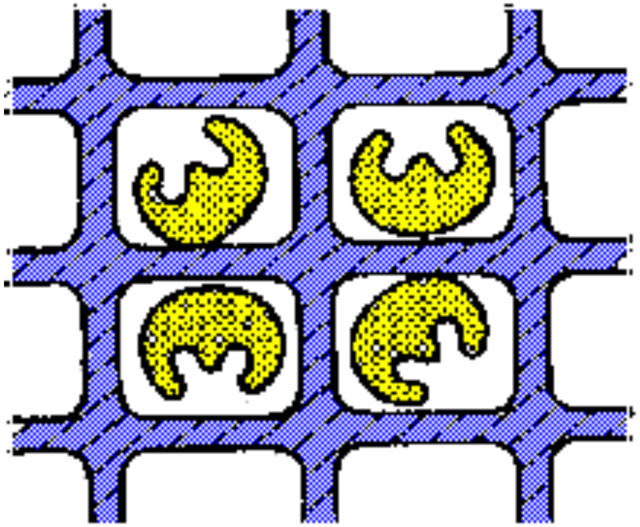
D. if (i > maxloops) then break

E. cJ = cond(J);

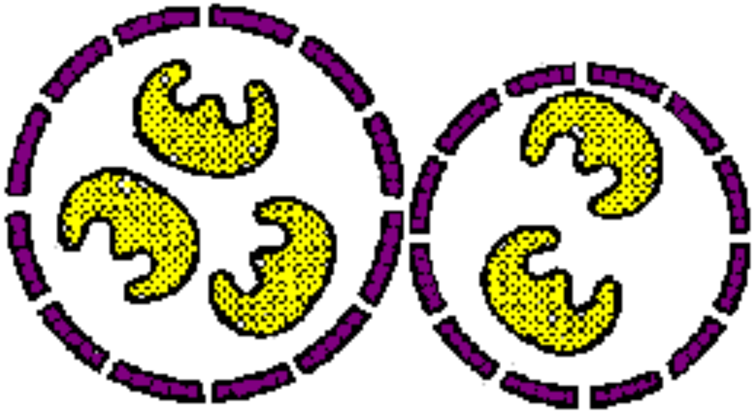
Immobilized Enzyme Systems

Entrapment

- Matrix Entrapment
- Membrane Entrapment (microencapsulation)



entrapped in a matrix



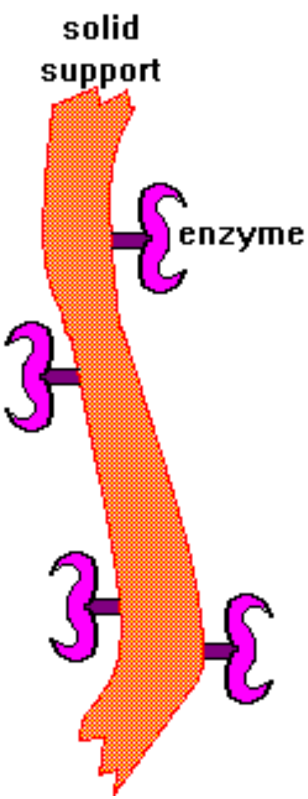
entrapped in droplets

Immobilized Enzyme Systems

Surface immobilization

According to the binding mode of the enzyme, this method can be further sub-classified into:

- Physical Adsorption: Van der Waals
  Carriers: silica, carbon nanotube, cellulose, etc.
  Easily desorbed, simple and cheap,
  enzyme activity unaffected.



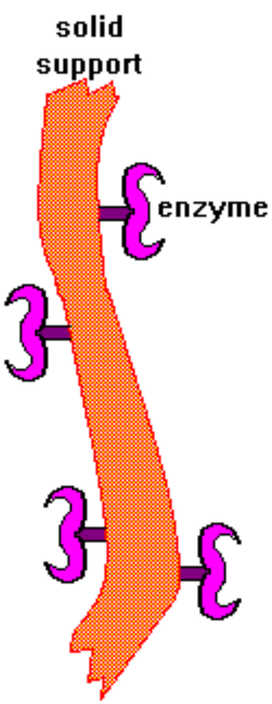
- Ionic Binding: ionic bonds
  Similar to physical adsorption.
  Carriers: polysaccharides and synthetic polymers having ion-exchange centers.

Immobilized Enzyme Systems

Surface immobilization

- <u>Covalent Binding</u>: covalent bonds

Carriers: polymers contain amino, carboxyl, sulfhydryl, hydroxyl, or phenolic groups.



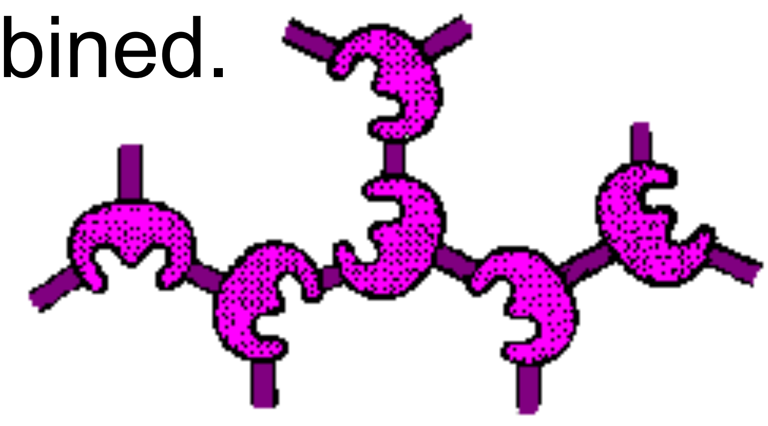
- Loss of enzyme activity
- Strong binding of enzymes

Immobilized Enzyme Systems

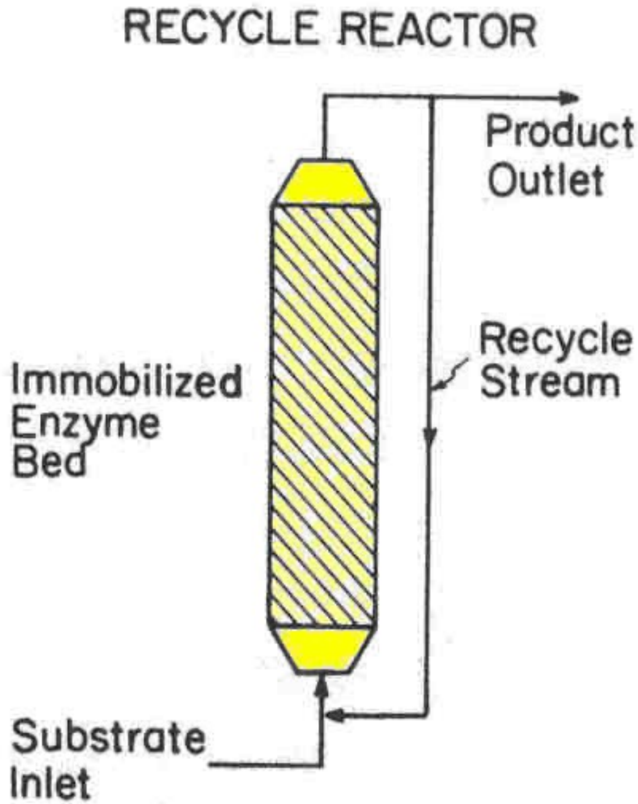
<u>Cross-linking</u> is to cross link enzyme molecules with each other using agents such as glutaraldehyde.

Features: similar to covalent binding.

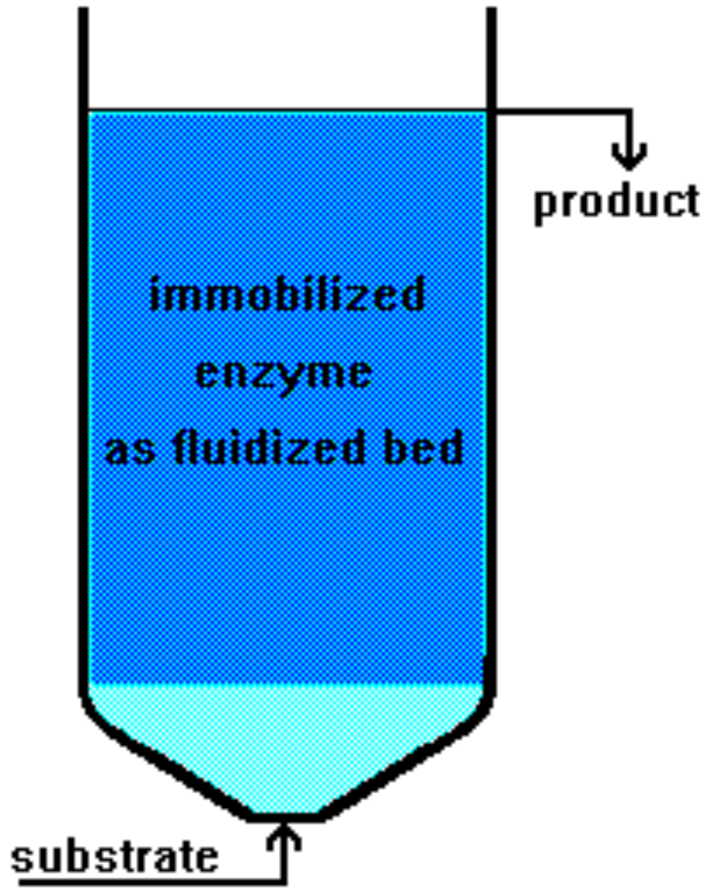
Several methods are combined.



Immobilized Enzyme Reactors



Recycle packed column reactor:
- allow the reactor to operate at high fluid velocities.



Fluidized Bed Reactor:

- a high viscosity substrate solution
- a gaseous substrate or product in a continuous reaction system
- care must be taken to avoid the destruction and
  decomposition of immobilized enzymes

Constrained Optimization

- So far, we have dealt with unconstrained optimization methods.
- Most of the time (except in regression problems) we have constraints on feasible solutions. How do we deal with this?
- One-dimension: In 1-D we may have inequality constraints (say $r > 0$).
- In this case we calculate the optimum in the interior and then compare to function on the boundary of feasible solution region.
- If F is lower on the boundary, then that point is the answer!

Multi-Dimensional Constraints

- If N-dimensional, the problem is much more complex.
- We can have either equality constraints or inequality constraints.
- Let's look at equality constraints first.
- In general, we seek

$$\min_{\mathbf{x}} F(\mathbf{x}) \text{ subject to } \mathbf{g}(\mathbf{x}) = \mathbf{0}$$

- The best way of treating this is to use the m constraints $\mathbf{g} = \mathbf{0}$ to eliminate m variables from $F(\mathbf{x})$!

Multidimensional Constraints

- We did this with the soup can example:

  $F(x_1,x_2) = 2\pi x_1^2 + 2\pi x_1 x_2$

  $\mathbf{g}(\mathbf{x}) = 2\pi x_1^2 x_2 - V = 0$

- We used this to eliminate x_2:

  $x_2 = V/2\pi x_1^2$

  $F = 2\pi x_1^2 + V/x_1$

- So F is now an unconstrained 1-D problem!

Multidimensional Constraints

- Usually you can't get away with this.
- One approach is using Lagrange multipliers!
- Let F* = F + λ * **g**

  where λ contains m multipliers for the m constraints **g**(**x**) = **0**.
- The optimization problem was the solution to

  grad(F) = 0

Lagrange Multipliers

- Here we have the augmented problem:

$$\nabla^* F^* = 0,$$

where

$$\nabla^* = \begin{pmatrix} \nabla_x \\ \nabla_\lambda \end{pmatrix}.$$

- This is beca

$$\mathbf{grad}_\lambda\, F^* = \mathbf{g}(\mathbf{x}) = \mathbf{0}$$

or just the equality constraints!

Inequality Constraints

- Now for inequality constraints.
- We have: $\mathbf{g}(\mathbf{x}) <= \mathbf{0}$
- We can convert <u>inequality</u> constraints to <u>equality</u> constraints by adding <u>slack variables</u>.
- Let $\mathbf{g} + \mathbf{s} = 0$

  where $s_i = x^2_{n+1} >= 0$

  (convenient choice ensuring that $s_i >= 0$)

We then treat the problem using Lagrange multipliers again!

Penalty Functions

- Finally, we look at penalty functions.
- We wish to have an unconstrained optimization problem. We can do this even with constraints in an artificial manner.
- Suppose we have:

$$\min_{\mathbf{x}} F(\mathbf{x}), \quad \mathbf{g}(\mathbf{x}) = \mathbf{0}$$

- We may define $F^*(\mathbf{x}) = F(\mathbf{x}) + P||\mathbf{g}(\mathbf{x})||^2$

  where P is a positive number.

Penalty Functions

- We proceed by obtaining the solution for x at moderate values of P, and then slowly increase it.
- P $\rightarrow$ infinity corresponds to g(x) $\rightarrow$ 0!
- This can also be used with inequality constraints through the use of slack variables.