ECE 330—Complete Course Notes

Course Instructor
Dr. Wolfgang Fink
Electrical and Computer Engineering, University of Arizona

Document Author
Matthias Guenther
Electrical and Computer Engineering, University of Arizona

Fall 2016

ECE 330 8/31/16

If you have a numerical problem to solve, see Numerical Recipes. It describes the existing methods for solving the problem, their derivations, issues with them, and more.

ECE 330

$$f'(x) = \frac{f(x+h)-f(x)}{h} + O(h)$$

Taylor series expansion:

$$f(x+h) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x)}{n!} h^n$$

(*) $$f(x+h) = f(x) + h f'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f^{(3)}(x) + \frac{h^4}{24} f^{(4)}(x) \ldots$$

(1) $$\Rightarrow \underbrace{f'(x) = \frac{f(x+h)-f(x)}{h}}_{\text{Two-point formula, error } O(h)} \underbrace{- \frac{h}{2} f''(x) - \frac{h^2}{6} f^{(3)}(x)}$$

(**) $$f(x-h) = f(x) - h f'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f^{(3)}(x) + \ldots$$

(2) $$\Rightarrow \underbrace{f'(x) = \frac{f(x)-f(x-h)}{h}}_{\text{Two-point formula, error } O(h)} \underbrace{+ \frac{h}{2} f''(x) - \frac{h^2}{6} f^{(3)}(x) + \frac{h^3}{24} f^{(4)}(x) \ldots}$$

Add (1) & (2); divide by 2:

$$\underbrace{f'(x) = \frac{f(x+h)-f(x-h)}{2h}}_{\text{Improved two-point formula, error } O(h^2)} \underbrace{- \frac{h^2}{6} f^{(3)}(x)}$$

$$(*) \Rightarrow f''(x) = \frac{f(x+h)-f(x)-hf'(x)}{\frac{h^2}{2}} - \frac{h}{3} f^{(3)}(x) - \frac{h^2}{12} f^{(4)}(x)$$

$$(**) \Rightarrow f''(x) = \frac{f(x-h)-f(x)+hf'(x)}{\frac{h^2}{2}} + \frac{h}{3} f^{(3)}(x) - \frac{h^2}{12} f^{(4)}(x)$$

Avg. both eqns.:

$$\underbrace{f''(x) = \frac{f(x+h)+f(x-h)-2f(x)}{h^2}}_{\text{Improved three-point formula, error } O(h^2)} \underbrace{- \frac{h^2}{12} f^{(4)}(x)}$$

ECE 330

Numerical Integration

Overarching theme

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{N} \omega_i f(x_i)$$

ω_i: weights; x_i: mesh pts. or particularly chosen pts.

Integrate $f(x)$ over $[a,b]$

divide $[a,b]$ into $n-1$ sub-intervals

mesh points: $a = x_1 < x_2 < x_3 < \ldots < x_n = b$

Note: x_i not necessarily equidistant!

func. values at these points:

$f_i = f(x_i) \quad i = 1, \ldots, n$

Trapezoidal Rule:

area $F_i = \frac{1}{2}(x_{i+1} - x_i)(f_{i+1} + f_i)$

$$\Rightarrow \int_a^b f(x)\,dx = I_{analyt.} \approx I_{TR} = F_1 + F_2 + F_3 + \ldots + F_{n-1}$$

If mesh points equidistant, then I_{TR} is easy to evaluate:

with $h = \frac{b-a}{n-1} \Rightarrow x_i = a + (i-1)h$ (mesh points)

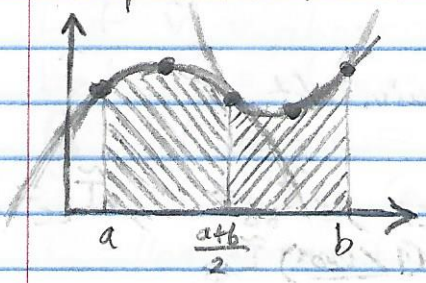
$$\left.\begin{aligned} F_i &= \tfrac{h}{2}(f_{i+1} + f_i) \\ F_{i+1} &= \tfrac{h}{2}(f_{i+2} + f_{i+1}) \\ &\vdots \qquad\qquad \vdots \\ F_{n-1} &= \tfrac{h}{2}(f_n - f_{n-1}) \end{aligned}\right\} \Rightarrow \boxed{I_{TR} = \tfrac{h}{2}f_1 + hf_2 + hf_3 + \ldots + hf_{n-1} + \tfrac{h}{2}f_n}$$

Error: $\left|I_{TR} - I_{analyt.}\right| \le \frac{h^2}{12}(b-a) \max_{a \le x \le b}\left(\left|f''(x)\right|\right)$

if $f'(x)$ exists

ECE 330

Simpson's Rule (SR)



Assume equidistant mesh points.

Consider integral over a parabola from -1 to 1:

$$\int_{-1}^{1} p(x)\,dx = \int_{-1}^{1} (ax^2+bx+c)\,dx = \left[\frac{a}{3}x^3+\frac{b}{2}x^2+cx+d\right]_{-1}^{1} = \frac{2a}{3}+2c$$

Parabola should intersect or touch integrand $f(x)$ at $x = -1, 0, +1$

$$\Rightarrow \begin{cases} f(-1) = p(-1) = a-b+c \\ f(0) = p(0) = c \\ f(+1) = p(+1) = a+b+c \end{cases} \Rightarrow \frac{2a}{3}+2c = \frac{f(-1)}{3}+\frac{4}{3}f(0)+\frac{f(1)}{3}$$

$$\Rightarrow \int_{-1}^{1} f(x)\,dx \approx \frac{2a}{3}+2c = \frac{f(-1)}{3}+\frac{4}{3}f(0)+\frac{f(1)}{3}$$

See D2L for derivation of end formula.

for $[\ell-h, \ell+h]$

$$\Rightarrow \int_{\ell-h}^{\ell+h} f(x)\,dx \approx h\left[\frac{f(\ell-h)}{3}+\frac{4}{3}f(\ell)+\frac{f(\ell+h)}{3}\right]$$

$$\int_{\ell-h}^{\ell+h} f(x)\,dx \xrightarrow{\text{shift by } h} \int_{\ell}^{\ell+2h} f(x)\,dx \approx h\left(\frac{f(\ell)}{3}+\frac{4}{3}f(\ell+h)+\frac{f(\ell+2h)}{3}\right)$$

$$\Rightarrow \int_a^b f(x)\,dx \approx h\left[\frac{f(a)}{3}+\frac{4}{3}f(a+h)+\frac{2}{3}f(a+2h)+\frac{4}{3}f(a+3h)+\frac{2}{3}f(a+4h)+\frac{4}{3}f(a+5h)+\ldots+\frac{f(b)}{3}\right]$$

$$\Rightarrow \int_a^b f(x)\,dx \approx \frac{h}{3}f_1+\frac{4h}{3}f_2+\underbrace{\frac{2h}{3}f_3+\frac{4h}{3}f_4+\ldots}_{\text{repeating portion}}+\frac{4h}{3}f_{n-1}+\frac{h}{3}f_n$$

Error: $|I_{SR} - I_{analyt.}| \leq$

ECE 330

$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$

Generalize TR and SR by interpolating function $f(x)$ between the points by a polynomial of nth degree and integrating it.

n	name	weights for single interval	weights for multiple intervals
1	TR	$\frac{h}{2}\ \frac{h}{2}$	$\frac{h}{2}\ [h]\ \frac{h}{2}$
2	SR	$\frac{h}{3}\ \frac{4h}{3}\ \frac{h}{3}$	$\frac{h}{3}\ \left[\frac{4h}{3}\ \frac{2h}{3}\right]\ \frac{4h}{3}\ \frac{h}{2}$
3	$\frac{3}{8}$ rule or Pulcherrima ("most beautiful")	$\frac{3h}{8}\ \frac{9h}{8}\ \frac{9h}{8}\ \frac{3h}{8}$	
4	Milne Rule	$\frac{14h}{45}\ \frac{64h}{45}\ \frac{24h}{45}\ \frac{64h}{45}\ \frac{14h}{45}$	
⋮	⋮	⋮	

⇒ Newton–Cotes Integration (NC)

$n \geq 7$ ⇒ NC becomes numerically unusable because of increasing rounding errors

⇒ NC rarely used (except TR, SR)

⇒ We need more effective higher integration rule.

Gauss–Legendre Integration (GLI)

if $f(x)$ can be well approximated by a power series ⇒ use GLI for a much higher accuracy

GLI operates with fewer points

⇒ better suited for multi-dimensional integrations and especially if integrand $f(x)$ is computationally expensive.

STOCHASTIC (later class?)

ECE 330

To understand GLI, we consider integral of straight line $g(x)$ over interval $[-1, 1]$.

TR delivers exact result:

$$\int_{-1}^{1} g(x)\,dx = \int_{-1}^{1} \underbrace{(bx+c)}_{g(x)}\,dx = 2c = \underbrace{g(-1)+g(1)}_{\text{2 points to evaluate integral}}$$

$$\left[\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)\right]$$

alternatively, $c = g(0)$

$$\Rightarrow \int_{-1}^{1} g(x)\,dx = \int_{-1}^{1} (bx+c)\,dx = 2c = \underbrace{2g(0)}_{\text{1 point}}$$

$\Rightarrow$ Single point suffices for exact integration if chosen correctly.

$\Rightarrow$ GLI is generalization of this idea to polynomials of higher degrees. By suitable choice of n mesh points and n integration weights, any polynomial up to degree $(2n-1)$ can be integrated exactly over $[-1,1]$

For GLI derivations

We define inner product in the space of functions in $[-1,1]$:

$$\langle f \mid g \rangle := \int_{-1}^{1} f(x) g(x)\,dx$$

"Dirac notation" in quantum mechanics

Define system of orthogonal polynomials with following properties:

$L_n(x) \equiv$ polynomial of degree n

$\langle L_m(x) \mid L_n(x) \rangle = 0$ for $m \neq n$

$\Rightarrow L_n(x)$ uniquely defined within a normalizing factor

usual normalization $\langle L_m(x) \mid L_n(x) \rangle = \delta_{mn} \cdot \frac{2}{2n+1}$

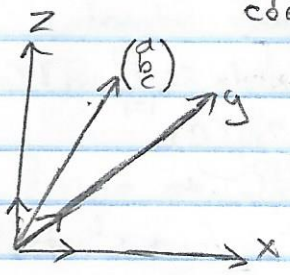
(δ_{mn}: Kronecker Delta)

ECE 330

Legendre polynomials form an orthogonal basis
$\Rightarrow$ all polynomials of degree n can be expanded in Legendre polynomials.

$$P_n = \sum_{i=0}^{n} \lambda_i L_i(x)$$

λ_i: expansion coeff. $L_i(x)$: Legendre poly. degree i



$$\hat{x} = \begin{pmatrix}1\\0\\0\end{pmatrix}, \quad \hat{y} = \begin{pmatrix}0\\1\\0\end{pmatrix}, \quad \hat{z} = \begin{pmatrix}0\\0\\1\end{pmatrix}$$

} form an orthogonal basis

$$\Rightarrow \begin{pmatrix}a\\b\\c\end{pmatrix} = a\hat{x} + b\hat{y} + c\hat{z}$$

$$P_n(x) = \lambda_0 L_0(x) + \lambda_1 L_1(x) + \ldots + \lambda_n L_n(x)$$

$$L_n(x) = L_n(x)$$

First 3 Leg. polynomials:

$$L_0(x) = 1$$
$$L_1(x) = x$$
$$L_2(x) = \frac{3}{2}x^2 - \frac{1}{2}$$

Generating formula for $L_n(x)$:

Rodriguez formula:

$$\boxed{L_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2-1)^n}$$

Important

Legendre polynomials $L_n(x)$ have exactly n unique roots in $(-1, 1)$.

No two Legendre polynomials have any shared roots!

ECE 330

GLI: n points can integrate polynomials $P_{2n-1}(x)$ exactly in $[-1, 1]$.

$$\int_{-1}^{1} P_{2n-1}(x)\,dx = \sum_{i=0}^{n} w_i P_{2n-1}(x_i)$$

(x_i: Roots of Legendre poly. $L_n(x)$)

This is what we want to show.

$$P_{2n-1}(x) = L_n(x) P_{n-1}(x) + q_{n-1}(x)$$

$$\frac{P_{2n-1}}{L_n} = P_{n-1} + \frac{q_{n-1}}{L_n} \quad \Big| \cdot L_n$$

$$\Rightarrow P_{2n-1} = L_n P_{n-1} + q_{n-1}$$

$$\int_{-1}^{1} P_{2n-1}(x)\,dx = \int_{-1}^{1} [L_n(x) P_{n-1}(x) + q_{n-1}(x)]\,dx$$

$$= \int_{-1}^{1} L_n(x) P_{n-1}(x)\,dx + \int_{-1}^{1} q_{n-1}(x)\,dx$$

$$= \langle L_n(x) \mid P_{n-1}(x) \rangle + \int_{-1}^{1} q_{n-1}(x)\,dx$$

$$= 0 + \int_{-1}^{1} q_{n-1}(x)\,dx$$

$$= \int_{-1}^{1} q_{n-1}(x)\,dx$$

ECE 330

$\int_{-1}^{1} P_{2n-1}(x)\,dx = ?$

$P_{2n-1}(x) = L_n(x) P_{n-1}(x) + q_{n-1}(x)$

$\Rightarrow \int_{-1}^{1} P_{2n-1}(x)\,dx = \int_{-1}^{1} q_{n-1}(x)\,dx$

At roots of $L_n(x)$: $x_1, \ldots, x_n$

$P_{2n-1}(x_i) = \underbrace{L_n(x_i)}_{0 \text{ at roots}} P_{n-1}(x_i) + q_{n-1}(x_i)$

$\Rightarrow P_{2n-1}(x_i) = q_{n-1}(x_i)$

We have n eqns. because we have n roots x_i.

$\Rightarrow$ Pol. $q_{n-1}(x)$ is completely determined

$\Rightarrow \int_{-1}^{1} q_{n-1}(x)\,dx$ is also determined

---

Expand $q_{n-1}(x)$ into Leg. polys.

$q_{n-1}(x) = \sum_{i=0}^{n-1} \alpha_i L_i(x)$

(α_i: expansion coeffs.; $L_i(x)$: Legendre pols.)

$$\int_{-1}^{1} q_{n-1}(x)\,dx = \int_{-1}^{1} \sum_{i=0}^{n-1} \alpha_i L_i(x)\,dx$$

$$= \sum_{i=0}^{n-1} \alpha_i \int_{-1}^{1} 1 \cdot L_i(x)\,dx$$

$$= \sum_{i=0}^{n-1} \alpha_i \int_{-1}^{1} L_0(x) L_i(x)\,dx = \sum_{i=0}^{n-1} \alpha_i \langle L_0 | L_i \rangle$$

$$= \alpha_0 \langle L_0 | L_0 \rangle$$

$$= \alpha_0 \frac{2}{2(0)+1} = \boxed{2\alpha_0}$$

$\int_{-1}^{1} P_{2n-1}(x)\,dx = \int_{-1}^{1} q_{n-1}(x)\,dx = 2\alpha_0$

How to determine α_0?

Since we know in principle $q_{n-1}(x)$, we also know function values of $q_{n-1}(x)$ at n roots of x_k

$\Rightarrow$ System of linear eqns.:

$$\sum_{i=0}^{n-1} L_i(x_k)\alpha_i = q_{n-1}(x_k) \quad \forall k = 1, \ldots, n$$

$\Rightarrow$ we use n eqns.

ECE 330

$$\sum_{i=0}^{n-1} L_{ki}\,\alpha_i = q_{n-1}(x_k) \quad \forall k = 1,\dots,n$$

Since L_n are <u>linearly independent</u> (because of orthogonality), $\rightarrow$ $(n \times n)$ matrix L_{ki} <u>invertible</u>!

$$\Rightarrow \alpha_k = \sum_{i=1}^{n} (L^{-1})_{ki}\, q_{n-1}(x_i) \quad \forall k = 0,\dots,n-1, \quad i = 1,\dots,n$$

(L^{-1}: inverse matrix)

$$\Rightarrow \alpha_0 = \sum_{i=1}^{n} (L^{-1})_{0i}\, q_{n-1}(x_i)$$

$$\boxed{\int_{-1}^{1} P_{2n-1}(x)\,dx = 2\alpha_0 = 2\sum_{i=1}^{n} (L^{-1})_{0i}\, q_{n-1}(x_i) = 2\sum_{i=1}^{n} (L^{-1})_{0i}\, P_{2n-1}(x_i)}$$

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

The inverse matrix L^{-1} is only determined by the properties of the Legendre polys. and is completely <u>independent</u> of the function to be integrated.

- equation has the form of an integration rule for n mesh points
- these n mesh pts. are roots of Leg. poly. $L_n(x)$
- the integration weights w_i are, aside from the factor 2, the top row of L^{-1}
- integration weights & roots of Leg. polys. are precalculated and published in tables

GLI for $f(x)$ on $[-1, 1]$:

$$\boxed{\int_{-1}^{1} f(x)\,dx \approx I_{GLI} = \sum_{i=1}^{n} \underbrace{2(L^{-1})_{0i}}_{w_i} f(\underbrace{x_i}_{\text{roots of } L_n})}$$

ECE 330

Significance of GLI:

- all polynomials up to degree $(2n-1)$ are integrated exactly
- analytical functions $f(x)$

ECE 330

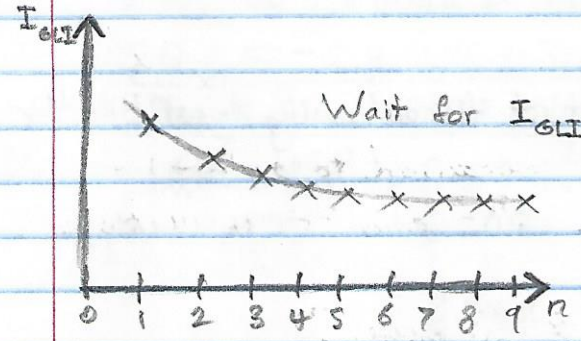
With linear transformation of integration variable, GLI is applicable to arbitrary finite interval $[a,b]$:

$$\int_a^b f(u)\,du = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{(b-a)x}{2}+\frac{b+a}{2}\right)dx$$

substitution:
$u: \frac{b-a}{2}x+\frac{b-a}{2}$
$\Rightarrow du = \frac{b-a}{2}dx$

$$\approx \boxed{\frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{(b-a)x_i}{2}+\frac{b+a}{2}\right)}$$

$$= I_{GLI}$$




```
I_GLI = 0;
for (i=1; i<=n; i++){
    I_GLI += w_i f( (b-a)/2 x_i + (b+a)/2 );
}
I_GLI *= (b-a)/2;
```

w_i, x_i: imported from table

ECE 330

Differential Equations (DE)

$m\ddot{x} = F$

general equation of motion has the form:

$\frac{d^2x}{dt^2} = f(x, \frac{dx}{dt}, t)$

($\frac{d^2x}{dt^2}$: acceleration)

$\Rightarrow$ 2nd order differential equation in one dim.

$\Rightarrow$ rewrite as a system of coupled first-order DEs.

introduce: $y_1(t) := x(t)$ location

$y_2(t) := \frac{dx(t)}{dt}$ velocity

$$\Rightarrow \boxed{\begin{aligned} \frac{dy_1(t)}{dt} &= y_2(t) \\ \frac{dy_2(t)}{dt} &= f(y_1, y_2, t) \end{aligned}}$$

This conversion can be generalized to multi-dim. oscillation eqns. and to eqns. for coupled oscillations.

How to solve (numerically) DEs of first order?

Euler Method (EM)

To understand EM, consider

$\frac{dy}{dt} = f(y, t)$

$\frac{y(t+h) - y(t)}{h} + O(h) = f(y, t)$

$\Rightarrow y(t+h) = y(t) + h f(y(t), t) + O(h^2)$

If we ignore error term $O(h^2)$, we get <u>recursion formula</u> allowing us to compute $y(t)$ from initial condition $y(0)$ in steps of h

$$y(t+h) = y(t) + h f(y(t), t) \qquad O(h^2) \text{ leading error}$$

For system of <u>coupled diff. eqns.</u>:

$$\frac{dy_i}{dt} = f_i(y_1, y_2, \ldots, y_n, t) \quad \forall i = 1, \ldots, n$$

$$\Rightarrow y_i(t+h) = y(t) + h f_i(y_1, y_2, \ldots, y_n, t) + O(h^2)$$

ECE 330

(See prev. page)

⇒ EM for for sys. of n coupled DEs of 1st order

$$\underbrace{y_i(t+h)}_{\text{future}} = \underbrace{y_i(t) + h f_i(y_1, \ldots, y_n, t)}_{\text{present}} + \underbrace{O(h^2)}_{\text{error term}}$$

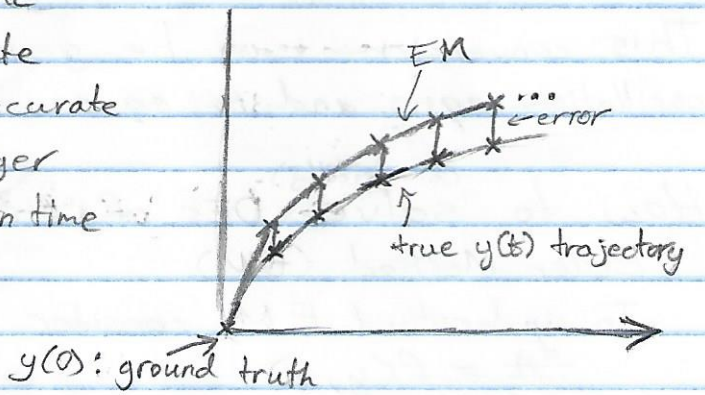
$\underbrace{\qquad\qquad\qquad\qquad}_{\text{practically used}}$

⇒ $y_i(t)$ can be computed approximately on a time mesh of width/stepsize h

Specify initial conditions $y_1(0), y_2(0), \ldots, y_n(0)$

Downsides

- EM not very accurate
- EM not very stable
- h large ⇒ inaccurate
- h small ⇒ more accurate but longer computation time



ECE 330

Euler Method (EM)

$$y_i(t+h) = \underbrace{y_i(t) + h f_i(y_1, y_2, \dots y_n, t)}_{\text{present}} + O(h^2)$$

+ initial conditions at $t=0$

$y_1(0), y_2(0), \dots, y_n(0)$

EM $\rightarrow O(h^2)$

Can we improve that?

Consider Taylor expansion:

assume $f(y,t)$ is differentiable a sufficient # of times with respect to y & t.

$$\Rightarrow y(t+h) = y(t) + h\frac{dy(t)}{dt} + \frac{h^2}{2}\frac{d^2y(t)}{dt^2} + O(h^3)$$

$$= y(t) + hf(y,t) + \frac{h^2}{2}\underbrace{\frac{df(y,t)}{dt}} + O(h^3)$$

we do not know this term

Use two-point formula for $\frac{df(y,t)}{dt}$

$$\frac{df(y,t)}{dt} = \frac{f(y(t+h), t+h) - f(y(t),t)}{h} + O(h)$$

$$\Rightarrow y(t+h) = y(t) + hf(y,t) + \frac{h^2}{2}\left[\frac{f(y(t+h), t+h) - f(y(t),t)}{h}\right]$$

present; future!; present

replace/approximate $y(t+h)$ with EM

$$y(t+h) = y(t) + hf(y(t), t) + O(h^2)$$

$$\Rightarrow y(t+h) = y(t) + hf(y,t) + \frac{h}{2}\left[f(y(t) + hf(y(t),t), t+h) - f(y(t),t)\right] + O(h^3)$$

$$\Rightarrow \boxed{y(t+h) = y(t) + h\left[\frac{f[y(t) + hf(y(t),t), t+h] + f(y(t),t)}{2}\right] + O(h^3)}$$

Improved Euler Method (IEM)

With following abbr.:

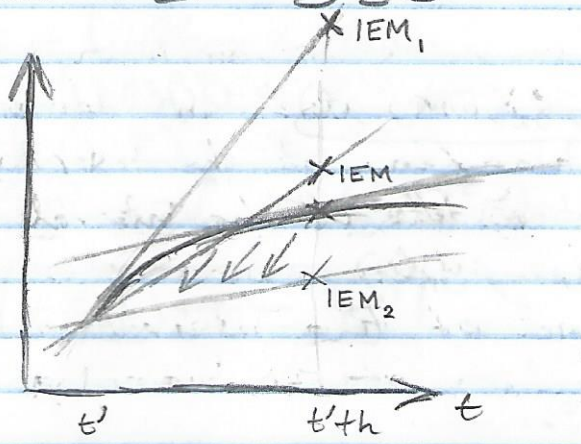
$k^{(1)} := f(y(t), t) \rightarrow$ slope at t

$k^{(2)} := f[y(t) + hk^{(1)}, t+h] \rightarrow$ approximation of slope at $(t+h)$

$\Rightarrow$ IEM: $y(t+h) = y(t) + \frac{h}{2}(k^{(1)} + k^{(2)}) + O(h^3)$

Instead of gradient of soln. curve $y(t)$ at mesh pt. t, we now use mean value of gradients at the mesh pts. t and $(t+h)$.

ECE 330



Immediate generalization to a set of coupled 1st-order DEs:

$$y(t) \rightarrow \vec{y}(t)$$

$$\Rightarrow \frac{dy_i(t)}{dt} = f_i(y_1, y_2, \cdots, y_n, t) \quad \forall i = 1, \cdots, n$$

$$k_i^{(1)} = f_i(y_1, y_2, \cdots, y_n, t)$$

$$k_i^{(2)} = f_i(y_1(t) + hk_1^{(1)}, y_2(t) + hk_2^{(1)}, \cdots, y_n(t) + hk_n^{(1)}, t+h)$$

$$\Rightarrow \boxed{y_i(t+h) = \underbrace{y_i(t) + \frac{h}{2}(k_i^{(1)} + k_i^{(2)})}_{\text{useful part}} + \underbrace{O(h^3)}_{\text{error term}} \quad \forall i = 1, \cdots, n}$$

Improved Euler Method for a set of coupled 1st-order DEs

Remember:

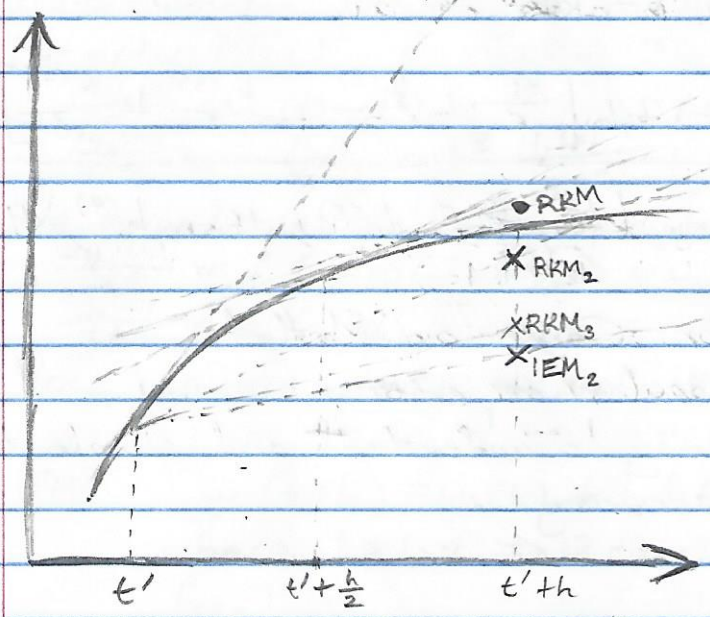
first calculate all $k_i^{(1)}$

then calculate all $k_i^{(2)}$

finally calculate all $y_i(t+h)$

ECE 330

IEM is superior to EM by one order in h.
Runge-Kutta Method (RKM) uses, in addition to gradients at the beginning/end of the interval, the gradient at the middle of the interval $(t+\frac{h}{2})$ with the values of the solution functions $y_i(t)$ at the middle and end of the interval suitably extrapolated from the values at the beginning of the interval.



RK4 (Runge-Kutta Method using 4 gradients) has error $O(h^5)$.

ECE 330

RK4:

Uses four gradients: $k_i^{(1)}$, $k_i^{(2)}$, $k_i^{(3)}$, $k_i^{(4)}$

$k_i^{(2)}$, $k_i^{(3)}$ are gradients at the middle of interval $(t+\frac{h}{2})$, calculated in different ways.

$$k_i^{(1)} = f_i(y_1(t), \ldots, y_n(t), t)$$

$$k_i^{(2)} = f_i(y_1(t) + \tfrac{h}{2}k_1^{(1)}, \ldots, y_n(t) + \tfrac{h}{2}k_n^{(1)}, t + \tfrac{h}{2})$$

$$k_i^{(3)} = f_i(y_1(t) + \tfrac{h}{2}k_1^{(2)}, \ldots, y_n(t) + \tfrac{h}{2}k_n^{(2)}, t + \tfrac{h}{2})$$

$$k_i^{(4)} = f_i(y_1(t) + hk_1^{(3)}, \ldots, y_n(t) + hk_n^{(3)}, t + h)$$

$\Rightarrow$ recursion formula of RK4:

$$y_i(t+h) = y_i(t) + h\left[\frac{k_i^{(1)}}{6} + \frac{k_i^{(2)}}{3} + \frac{k_i^{(3)}}{3} + \frac{k_i^{(4)}}{6}\right] + O(h^5)$$

Calculate $k_i^{(1)}$, then $k_i^{(2)}$, then $k_i^{(3)}$, then $k_i^{(4)}$, then $y_i(t+h)$

So-called Predictor-Corrector Methods

recently more popular/superior

however, RK4 easy to understand and simple to program, with good accuracy.

ECE 330

Application of EM, IEM, and RK4

Consider <u>anharmonic</u> forced oscillation

general eqn. of motion:

$$M\frac{d^2x}{dt^2} = -A|x|^B\frac{x}{|x|} + C\cos(\omega t)$$

M: mass; A: const.; $|x|^B$: potential; $\frac{x}{|x|}$: unit vector; C: const.

$-A|x|^B\frac{x}{|x|}$: anharmonic restoring force (harmonic if B = 1)

$C\cos(\omega t)$: driving force

Transformation of DE of 2nd order into set of coupled 1st-order DEs:

$$\frac{dy_1(t)}{dt} = y_2(t)$$

$$\frac{dy_2(t)}{dt} = -\frac{A}{M}|y_1(t)|^B\frac{y_1(t)}{|y_1(t)|} + \frac{C}{M}\cos(\omega t)$$

We now have two first-order coupled DE's.

$$\frac{dy_1(t)}{dt} = f_1(y_1(t), y_2(t), t) = y_2(t)$$

$$\frac{dy_2(t)}{dt} = f_2(y_1(t), y_2(t), t) = -\frac{A}{M}|y_1(t)|^B\frac{y_1(t)}{|y_1(t)|} + \frac{C}{M}\cos(\omega t)$$

Now we are ready to apply EM, IEM, RK4

EM: $y_i(t+h) = y_i(t) + hf_i(y_1, y_2, \ldots, y_n, t)$

here, $i = 1, 2 \Rightarrow$ 2 eqns.

[...]

IEM: $y_i(t+h) = y_i(t) + \frac{h}{2}(k_i^{(1)} + k_i^{(2)})$

with $k_i^{(1)} = f_i(y_1, \ldots, y_n, t)$

$k_i^{(2)} = f_i(y_1 + hk_1^{(1)}, \ldots, y_n + hk_n^{(1)}, t+h)$

again, $i = 1, 2$

$\Rightarrow k_1^{(1)} = f_1 = y_2(t)$ $\quad k_2^{(1)} = f_2 = -\frac{A}{M}|y_1(t)|^B\frac{y_1(t)}{|y_1(t)|} + \frac{C}{M}\cos(\omega t)$

$k_1^{(2)} = f_1(y_1, hk_1^{(1)}, y_2 + hk_2^{(1)}, t+h)$

$= y_2(t) + hk_2^{(1)}$

$k_2^{(2)} = f_2(y_1 + hk_1^{(1)}, y_2 + hk_2^{(1)}, t+h) = -\frac{A}{M}|y_1(t) + hk_1^{(1)}| \cdot [\ldots]$

$[\ldots]\frac{y_1(t) + hk_1^{(1)}}{|y_1(t) + hk_1^{(1)}|} + \frac{C}{M}\cos(\omega(t+h))$

$\Rightarrow y_1(t+h) = y_1(t) + \frac{h}{2}(k_1^{(1)} + k_2^{(1)})$

$y_2(t+h) = y_2(t) + \frac{h}{2}(k_2^{(1)} + k_2^{(1)})$

} initial conditions

ECE 330

Application of EM, IEM, and RK4

Consider <u>anharmonic</u> forced oscillation

general eqn. of motion:

$$M\frac{d^2x}{dt^2} = -A|x|^B\frac{x}{|x|} + C\cos(\omega t)$$

M: mass; A: const.; $|x|^B$: potential; $\frac{x}{|x|}$: unit vector; C: const.

$-A|x|^B\frac{x}{|x|}$: anharmonic restoring force (harmonic if B = 1)

$C\cos(\omega t)$: driving force

Transformation of DE of 2nd order into set of coupled 1st-order DEs:

$$\frac{dy_1(t)}{dt} = y_2(t)$$

$$\frac{dy_2(t)}{dt} = -\frac{A}{M}|y_1(t)|^B\frac{y_1(t)}{|y_1(t)|} + \frac{C}{M}\cos(\omega t)$$

We now have two first-order coupled DEs,

$$\frac{dy_1(t)}{dt} = f_1(y_1(t), y_2(t), t) = y_2(t)$$

$$\frac{dy_2(t)}{dt} = f_2(y_1(t), y_2(t), t) = -\frac{A}{M}|y_1(t)|^B\frac{y_1(t)}{|y_1(t)|} + \frac{C}{M}\cos(\omega t)$$

Now we are ready to apply EM, IEM, RK4

EM: $y_i(t+h) = y_i(t) + hf_i(y_1, y_2, \ldots, y_n, t)$

here, $i = 1, 2 \Rightarrow$ 2 eqns.

[...]

IEM: $y_i(t+h) = y_i(t) + \frac{h}{2}(k_i^{(1)} + k_i^{(2)})$

with $k_i^{(1)} = f_i(y_1, \ldots, y_n, t)$

$k_i^{(2)} = f_i(y_1 + hk_1^{(1)}, \ldots, y_n + hk_n^{(1)}, t+h)$

again, $i = 1, 2$

$\Rightarrow k_1^{(1)} = f_1 = y_2(t)$ $\quad k_2^{(1)} = f_2 = -\frac{A}{M}|y_1(t)|^B\frac{y_1(t)}{|y_1(t)|} + \frac{C}{M}\cos(\omega t$

$k_1^{(2)} = f_1(y_1, hk_1^{(1)}, y_2 + hk_2^{(1)}, t+h)$

$= y_2(t) + hk_2^{(1)}$

$k_2^{(2)} = f_2(y_1 + hk_1^{(1)}, y_2 + hk_2^{(1)}, t+h) = -\frac{A}{M}|y_1(t) + hk_1^{(1)}| \cdot [\ldots]$

$[\ldots]\frac{y_1(t) + hk_1^{(1)}}{|y_1(t) + hk_1^{(1)}|} + \frac{C}{M}\cos(\omega(t+h))$

$\Rightarrow y_1(t+h) = y_1(t) + \frac{h}{2}(k_1^{(1)} + k_2^{(1)})$

$y_2(t+h) = y_2(t) + \frac{h}{2}(k_2^{(1)} + k_2^{(1)})$

} initial conditions

ECE 330

RK4:

$$y_i(t+h) = y_i(t) + h\left(\frac{k_i^{(1)}}{6} + \frac{k_i^{(2)}}{3} + \frac{k_i^{(3)}}{3} + \frac{k_i^{(4)}}{6}\right)$$

with $k_i^{(1)} = f_i(y_1, y_2, t)$

$k_i^{(2)} = f_i\left(y_1 + \frac{h}{2}k_1^{(1)}, y_2 + \frac{h}{2}k_2^{(1)}, t+h\right)$

$k_i^{(3)} = f_i\left(y_1 + \frac{h}{2}k_1^{(2)}, y_2 + \frac{h}{2}k_2^{(2)}, t+h\right)$

$k_i^{(4)} = f_i\left(y_1 + hk_1^{(3)}, y_2 + hk_2^{(3)}, t+h\right)$

again, $i = 1, 2$

$k_1^{(1)} = f_1 = y_2(t)$

$k_2^{(1)} = f_2 = -\frac{A}{M}|y_1(t)|^B \frac{y_1(t)}{|y_1(t)|} + \frac{C}{M}\cos(\omega t)$

$k_1^{(2)} = f_1\left(y_1 + \frac{h}{2}k_1^{(1)}, y_2 + \frac{h}{2}k_2^{(1)}, t + \frac{h}{2}\right) = y_2(t) + \frac{h}{2}k_2^{(1)}$

ECE 330

Computation of Electric Fields

Method of Successive Over-Relaxation (Liebmann Method)

Electrostatics & magnetostatics governed by Maxwell's Equations.

Normally, one solves Maxwell's Equations for highly specialized (ideal) examples: Plate condenser, spherical condenser, point charge in front of a conducting plane, infinitely long coil, etc.

It's hard to calculate electric field configurations for more complex/realistic structures.
↳ Computer simulation can help.

Take-Home Message:
- Solving boundary problems in PDEs
- Discretization of differential operators
- Clever use of coordinate transformations & symmetries
- Finite element method (i.e., soln. on a 2D grid) with successive over-relaxation (Liebmann method)

Gauss's Law: $\vec{\nabla}\cdot\vec{E} = \frac{\rho}{\varepsilon_0}$ (ρ ← charge density, ε_0 ← electric const.)

$\vec{\nabla}\cdot$ = del operator; $\vec{\nabla}\cdot\vec{E}$ = div $\vec{E}$

In a vacuum: $\rho = 0 \Rightarrow \vec{\nabla}\cdot\vec{E} = 0$

$\vec{E} = -\vec{\nabla}\phi_{\text{electric pot.}}$

$\vec{F} = -\vec{\nabla}\Phi_{\text{pot. energy}}$

$\Rightarrow -\vec{\nabla}\vec{\nabla}\phi_{\text{electric}} = 0$

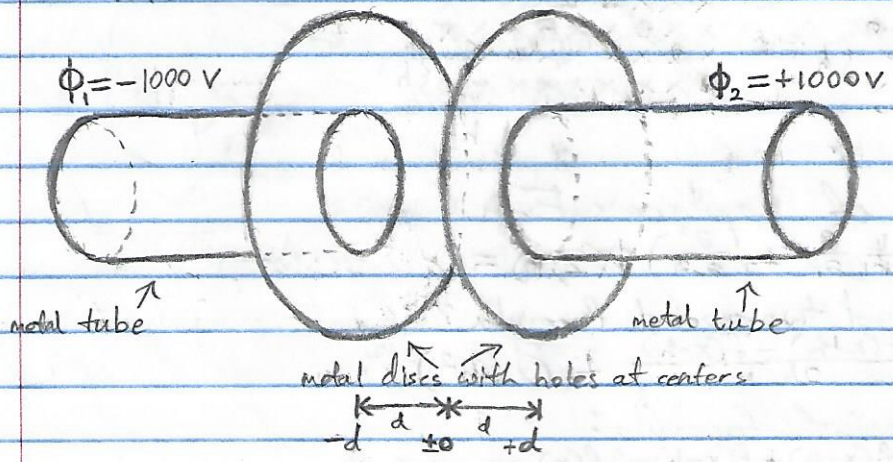
$\Rightarrow \boxed{\Delta\phi_{\text{electric}} = 0}$ Laplace's Eqn.

$$\vec{\nabla}^2 = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix} = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} = \Delta$$

Δ = Laplace operator

ECE 330

Electric Field Lens:



- Close volume
- Take advantage of <u>axial symmetry</u> of lens configuration
  ⇒ use cylindrical coordinates (φ, r, z)
  axis of tube is z axis
  r is dist. from z axis
  φ is azimuth angle around z axis

Laplace's equation in cylindrical coordinates:

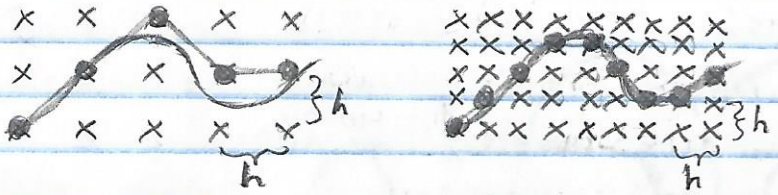
$$\Delta\phi = \frac{\partial^2\phi}{\partial z^2} + \frac{1}{r}\frac{\partial\phi}{\partial r} + \frac{\partial^2\phi}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2\phi}{\partial\varphi^2} = 0$$

For our system, $\Delta\phi = \left(\frac{\partial^2}{\partial z^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{\partial^2}{\partial r^2}\right)\phi(z,r) = 0$ for $r > 0$
Furthermore, the system is symmetrical (even) top-to-bottom and symmetrical (odd) left-to-right.
⇒ Only one quadrant needs to be calculated to find all others.

Next step: discretization of Laplace's equation
Approximate $\phi(z,r)$ on a 2D grid of pts. with a const. <u>mesh width h</u> (grid constant).
⇒ $\phi(z,r) \rightarrow$ matrix $(\phi_{i,k})$ of function values on grid pts.
$z_i := ih$, $r_k := kh$ ⇒ $\phi_{i,k} = \phi(z_i, r_k)$


} polygonal approximation necessary in some cases

ECE 330



Discretization of Laplace's Eqn.

$$\Delta\phi = \left(\frac{\partial^2}{\partial z^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{\partial^2}{\partial r^2}\right)\phi(z,r) = 0$$

Recall improved two-point formula:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

and three-point formula:

$$f''(x) = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} + O(h^2)$$

$$\Rightarrow \Delta\phi = \frac{1}{h^2}\Big[\underbrace{\phi(z,r+h) + \phi(z,r-h) - 2\phi(z,r)}_{\frac{\partial^2}{\partial r^2}} + \underbrace{\phi(z+h,r) + \phi(z-h,r) - 2\phi(z,r)}_{\frac{\partial^2}{\partial z^2}}\Big] + \frac{1}{r}\cdot\frac{1}{2h}\left[\phi(z,r+h) - \phi(z,r-h)\right] + \cancel{O(h^2)} = 0$$

ignore in following

Multiply by h^2 and ignore $O(h^2)$:

$$\Rightarrow \boxed{\phi_{i,k+1} + \phi_{i,k-1} + \phi_{i+1,k} + \phi_{i-1,k} - 4\phi_{i,k} + \frac{1}{2k}\left(\phi_{i,k+1} - \phi_{i,k-1}\right) = 0}$$

Discretized Laplace Equation for $k > 0$

Case: $k = 0$ ($r = 0 \Rightarrow$ z-axis)

$$\Delta\phi(x,y,z) = \left[\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right]\phi(x,y,z)$$

Apply three-point formula

$\frac{1}{h^2}[\ldots]$

$$\underbrace{\phi^{cart}(+h,0,z) = \phi^{cart}(-h,0,z) = \phi^{cart}(0,+h,z) = \phi^{cart}(0,-h,z)}_{= \phi^{cylind.}(z,h)}$$

$\Rightarrow$ for $x = y = 0$ ($\Leftrightarrow$ z-axis)

$$\frac{1}{h^2}\left[\phi(+h,0,z) + \phi(-h,0,z) + \phi(0,+h,z) + \phi(0,-h,z) + \phi(0,0,z+h) + \phi(0,0,z-h) - 6\phi(0,0,z)\right] + O(h^2) = 0$$

ECE 330

$\Rightarrow \frac{1}{h^2}\left[\phi_{i,1} + \phi_{i,1} + \phi_{i,1} + \phi_{i,1} + \phi_{i+1,0} + \phi_{i-1,0} - 6\phi_{i,0}\right] = 0$

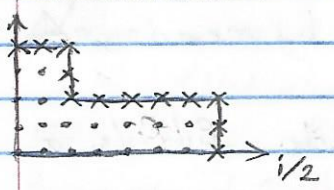
multiply by h^2 and ignore $O(h^2)$

$\Rightarrow$ $\boxed{4\phi_{i,1} + \phi_{i+1,0} + \phi_{i-1,0} - 6\phi_{i,0} = 0}$

Axis formula for $k=0$

$$\boxed{\phi_{i,k+1} + \phi_{i,k-1} + \phi_{i+1,k} + \phi_{i-1,k} - 4\phi_{i,k} + \frac{1}{2k}\left(\phi_{i,k+1} - \phi_{i,k-1}\right) = 0}$$

for $k>0$

$\Rightarrow$ 

$\Rightarrow$ This constitutes a set of linear equations in $\phi_{i,k}$ to solve Laplace's eqn. for interior grid pts. (not boundary pts.) of the electric lens.

$\Rightarrow$ In principle, matrix $(\phi_{i,k})$ has a high dimension because we calculate a lot of points.

$\Rightarrow$ Gaussian elimination is not a good choice because of high dimensionality and sparsity.

Sys. of lin. eqns.

$$\left.\begin{aligned} a_{00}x_0 + a_{01}x_1 + \dots + a_{0n}x_n &= b_0 \\ a_{10}x_0 + a_{11}x_1 + \dots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{n0}x_0 + a_{n1}x_1 + \dots + a_{nn}x_n &= b_n \end{aligned}\right\} \quad A\begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_0 \\ \vdots \\ b_n \end{bmatrix}$$

Matrix is highly sparse because points depend only on nearest neighbors. Therefore, Gaussian elimination would not succeed (in addition to taking far too long).

ECE 330

If Laplace operator were depicted via a matrix, it would be very sparse (mostly 0s).
⇒ Gauss Elimination not suitable for this
⇒ For such problems we use Method of Successive Over-Relaxation (Liebmann Method) (abbr. here as MSO)

System of lin. eqns. is solved iteratively:

1) Postulate initial approx. soln.
2) Calculate a better approximate soln. by means of an update formula.
3) Check whether improved approx. soln. fulfills a quality/exit criterion.
   a) If NO, replace original approx. soln. with the improved approx. soln.
      Repeat from 2).
   b.) If YES, improved soln. is final soln. ⇒ DONE.

In our electric-lens case:

1) Assume reasonable electric potential values for the interior region of the lens. Boundary points are given by fixed boundary conditions and therefore are not updated by MSO.
   Assume values around or close to boundary conditions to reduce iterations.
2) Calculate electric potential U for each grid point (i,k), one after another, from electric potential values at neighboring points (only nearest neighbors).

$k>0 \Rightarrow U = \phi_{i,k} = \frac{1}{4}\left[\underbrace{\phi_{i,k+1}}_{\text{North}} + \underbrace{\phi_{i,k-1}}_{\text{South}} + \underbrace{\phi_{i+1,k}}_{\text{East}} + \underbrace{\phi_{i-1,k}}_{\text{West}}\right] + \frac{1}{8k}\left(\underbrace{\phi_{i,k+1}}_{\text{North}} - \underbrace{\phi_{i,k-1}}_{\text{South}}\right)$

$k=0 \Rightarrow U = \phi_{i,0} = \frac{1}{6}\left[\underbrace{\text{East}}_{\text{East}} + \underbrace{\text{West}}_{\text{West}} + 4\,\underbrace{\text{North}}_{\text{North}}\right]$

Improved approx. soln. at pt. (i,k) calculated by means of the following formula:

$$\phi_{i,k}^{new} = \phi_{i,k}^{old} + w\,(U - \phi_{i,k}^{old})$$

w: over-relaxation parameter $\quad 0 \le w < 2$

Hopfield Attractor Network ?

ECE 330

$w = 0$ → valid but pointless (no change)

$0 < w < 1$ → slower convergence or relaxation toward solution

$w = 1$ → replaces old $\phi_{i,k}$ with the updated value U

$1 < w < 2$ → accelerates relaxation/convergence process

⇒ <u>successive over-relaxation</u>

$w \geq 2$ → always leads to instabilities

⇒ will not converge

Note: $w \approx 1.2$ tends to be a good value to choose.

(Not always, but often)

Choose w for fastest convergence. The solution will not change.

<u>Important</u>: The order in which all <u>interior</u> grid points are updated is <u>arbitrary</u>.

3) Exit criterion:

Ideal case: $\phi_{i,k}^{new} = \phi_{i,k}^{old}$ $\forall$ grid points (i,k)

Then, matrix of electric potential values $\phi_{i,k}$ satisfies the discretized Laplace equation with the specified boundary conditions along the lens surface.

More realistically: $|\phi_{i,k}^{new} - \phi_{i,k}^{old}| \leq \varepsilon$ $\forall (i,k)$

ε: user-defined threshold

e.g., $\varepsilon = 10^{-4}$ V $\quad \varepsilon \ll 1$

If w is too large (but still <2), oscillations can occur

⇒ higher number of iterations toward convergence

$w \geq 2$ always fails

w must be determined empirically — difficult to predict

Start with small values and increase or do bisection search

ECE 330

Workings of MSO

U calculated from elec. pot. vals. at neighboring pts.

Information propagates from boundaries inward.

Updating outer points (not boundary pts, though) before inner points can reduce iterations because information only propagates inward by one mesh width each iteration.

Information propagation is similar to a "diffusion process."

$w > 1$ accelerates "diffusion process."

Solving Systems of Linear Equations

- Gaussian Elimination (GE)
- Gauss-Jordan Elimination (GJE)

Gaussian Elimination

Method of successive elimination of the unknowns

Example:

$$3x - 4y + z = 1 \quad (\text{I})$$
$$2x + y - 2z = 3 \quad (\text{II})$$
$$x + 2y - z = 5 \quad (\text{III})$$

first, $(\text{I})/3 \Rightarrow x - \frac{4}{3}y + \frac{1}{3}z = \frac{1}{3}$ $(\tilde{\text{I}})$

then, subtract $2 \cdot (\tilde{\text{I}})$ from (II) $\Rightarrow$ $\frac{11}{3}y + \frac{4}{3}z = \frac{7}{3}$ $(\tilde{\text{II}})$

subtract $1 \cdot (\tilde{\text{I}})$ from (III) $\Rightarrow$ $\frac{10}{3}y - \frac{4}{3}z = \frac{14}{3}$ $(\tilde{\text{III}})$

next, $(\tilde{\text{II}})/\frac{11}{3} \Rightarrow y + \frac{4}{11}z = \frac{7}{11}$ $(\tilde{\tilde{\text{II}}})$

then, subtract $\frac{10}{3}$ $(\tilde{\tilde{\text{II}}})$ from $(\tilde{\text{III}}) \Rightarrow$

ECE 330

Consider generalization to negations of N unknowns:

$x_1, \ldots, x_n$:

$$\left.\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + \ldots + a_{1,N}x_N &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + \ldots + a_{2,N}x_N &= b_2 \\ \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots & \\ a_{N,1}x_1 + a_{N,2}x_2 + a_{N,3}x_3 + \ldots + a_{N,N}x_N &= b_N \end{aligned}\right\} A\vec{x} = b$$

General procedure:

elimination steps:
- divide 1st eqn by $a_{1,1}$
- subtract $a_{2,1}$ times init. result from 2nd eqn
- subtract $a_{3,1}$ times init. result from 3rd eqn
- continue until we have $N-1$ eqns in $N-1$ variables $x_2, \ldots, x_N$
- eliminate all eqns and xs except the last in this manner

backsubstitution steps:
- solve last eqn for x_N (all other xs removed)
- plug x_N into 2nd-to-last eqn to find x_{N-1}
- repeat for all other unknowns $x_{N-2}, x_{N-3}, \ldots, x_2, x_1$

This ignores several issues!

a.) If an unknown doesn't appear in an eqn. (coeff. = 0), use another coefficient (one that isn't 0).

Partial pivoting: choose the largest coeff. in the column of the variable you're eliminating.

b.) In which order should unknowns be eliminated? Instead of looking only at the next column, find the largest coefficient in the entire 2D array and use that column.

⇒ This is called complete pivoting. It's generally not worth the large search. Partial pivoting, however, is worth the effort.

Partial pivoting ↔ "reordering" the equations

Complete pivoting ↔ "reordering" both equations and unknowns.

ECE 330

c.) Scaling:

pivoting $\leftrightarrow$ "pick the largest element"

however, eqn. #i $\Leftrightarrow$ C · eqn. #i

(C: scalar)

It can make sense to scale equations to similar magnitudes.

d.) Rank of the sys. of linear eqns.

During GE process we might encounter rest of coeffs. in a column are zero, or we find a row or rows of zeros blocking us from proceeding with the method

$\Rightarrow$ means rank of matrix of all coeffs. is NOT N but lower!

$\Rightarrow$ system is underdefined, i.e., more unknowns than defining eqns.

Gauss-Jordan Elimination (GJE)

Introduced by Wilhelm Jordan in 1887 (not in 1920 as listed in Kreyszig (he lived 1842-1899).

GJE variant of GE in which back-substitution is avoided by additional computations that reduce the matrix to diagonal form as opposed to triangular form

$\Rightarrow$ requires more operations than back-subst. in GE

$\Rightarrow$ mthd is disadvantageous for solving linear eqn. systems

$\Rightarrow$ suffers from the same pitfalls as GE

BUT it can be used for matrix inversion!

ECE 330

After successive elimination, we have

$$\begin{aligned} x_1 + x + x + \ldots + x + x &= x \\ x_2 + x + \ldots + x + x &= x \\ x_3 + \ldots + x + x &= x \\ &\vdots \\ x_{N-1} + x &= x \\ x_N &= x \end{aligned}$$

triangular form

Instead of back-subst, use last eqn. to eliminate x_N in the first $N-1$ eqns., then use second-to-last eqn. to eliminate x_{N-1} in the first $N-2$ eqns., etc.

$\Rightarrow$ This will result in a diagonal matrix with the solns. explicitly given!

Matrix Inversion via GJE:

Rewrite matrix $(N \times N)$ to invert as a $(N \times 2N)$ matrix as follows:

$$\left[\begin{array}{cccc|cccc} a_{1,1} & a_{1,2} & \cdots & a_{1,N} & 1 & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,N} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} & 0 & 0 & \cdots & 1 \end{array}\right]$$

$(N \times N)$ — Identity matrix $(N \times N)$

Apply steps of GJE to transform first N columns into identity matrix $(N \times N)$, but apply the same steps to the identity matrix.

We need to recognize that each step of GJE can be written as a matrix applied to the matrix to be solved.

ECE 330

We can write all these steps 1, ..., N as a single matrix:

$$\underline{\underline{C}} = \underline{\underline{C}}_N \cdot \underline{\underline{C}}_{N-1} \cdot \underline{\underline{C}}_{N-2} \cdot \ldots \cdot \underline{\underline{C}}_3 \cdot \underline{\underline{C}}_2 \cdot \underline{\underline{C}}_1$$

$$\underline{\underline{C}} = \prod_{i=N}^{1} \underline{\underline{C}}_i$$

Examples

$$\underline{\underline{A}} = \begin{pmatrix} 1 & 1 & 4 \\ -1 & 0 & 1 \\ 1 & -1 & -5 \end{pmatrix}$$

Rewrite as $(N \times 2N)$

$$\begin{array}{ll} & \left(\begin{array}{ccc|ccc} 1 & 1 & 4 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 & 1 \\ 1 & -1 & -5 & 0 & 0 & 1 \end{array}\right) \begin{array}{l} (I) \\ (II) \\ (III) \end{array} \\ +(I) & \\ -(I) & \end{array}$$

$\underline{\underline{A}}$ $\quad$ $\underline{\underline{I}}$

Shared GE/GJE steps

$$\begin{array}{ll} & \left(\begin{array}{ccc|ccc} 1 & 1 & 4 & 1 & 0 & 0 \\ 0 & 1 & 5 & 1 & 1 & 0 \\ 0 & -2 & -9 & -1 & 0 & 1 \end{array}\right) \\ +2(II) & \\ -4(III) & \left(\begin{array}{ccc|ccc} 1 & 1 & 4 & 1 & 0 & 0 \\ 0 & 1 & 5 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 2 & 1 \end{array}\right) \\ -5(III) & \end{array}$$

Triangular (GE)

GJE-only steps

$$\begin{array}{ll} -(II) & \left(\begin{array}{ccc|ccc} 1 & 1 & 0 & -3 & -8 & -4 \\ 0 & 1 & 0 & -4 & -9 & -5 \\ 0 & 0 & 1 & 1 & 2 & 1 \end{array}\right) \\ & \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & -4 & -9 & -5 \\ 0 & 0 & 1 & 1 & 2 & 1 \end{array}\right) \end{array}$$

Diagonal (GJE)

$\underline{\underline{I}}$ $\quad$ $\underline{\underline{A}}^{-1}$

ECE 330

Curve fitting

$f(x_i) \approx y_i$

Type of function suggested by nature of data-generating process (e.g. linear, quadratic, polynomial, exponential, trigonometric, etc.)

Dangers

1) Overfitting data
2) Fitting may suggest that the data-generating process follows the wrong class of functions.

(Make the fit as simple as possible and as complex as necessary.)

3) Outliers skew fitting procedure

4) Not enough data (undersampling the process)
   → want to have as many data points as feasible
5) Ensure a "good" range (in x/t/etc.) of data, or else the fit may be based on a small-scale "point cloud" or have incorrect end behavior.

Linear Regression

fit $f(x) = a + bx$ to data (x_i, y_i) for $i = 1, \dots, M$

need to minimize quadratic error with respect to a, b: $F(a,b) = \sum_{i=1}^{M} (f(x_i) - y_i)^2$

$$\frac{\partial F(a,b)}{\partial a} = 2 \sum_{i=1}^{M} (a + bx_i - y_i) = 0$$

$$\frac{\partial F(a,b)}{\partial b} = 2 \sum_{i=1}^{M} (a + bx_i - y_i) x_i = 0$$

$$\begin{aligned} aM + b\sum_{i=1}^{M} x_i &= \sum_{i=1}^{M} y_i \\ a\sum_{i=1}^{M} x_i + b\sum_{i=1}^{M} x_i^2 &= \sum_{i=1}^{M} x_i y_i \end{aligned}$$

} So-called normal equations
Solve for unknowns a, b

ECE 330

Polynomial Curve Fitting

Fit polynomial of degree $N \ll M$ [...]

N = # of data points
bare minimum $N+1$
generally at least $10 \cdot N$
the more the better

[...] to M measurements / observations / data points

$$p(x) = \underbrace{a_0 + a_1 x + a_2 x^2 + \ldots + a_{N-1} x^{N-1} + a_N x^N}_{(N+1) \text{ coefficients } a_i}$$

Sum of the squared differences:

$$\sum_{i=1}^{M} [p(x_i) - y_i]^2$$

($p(x_i)$: approximation; y_i: measured/actual data point)

$(N+1)$ partial derivatives to minimize $F(\ldots)$

$$\frac{\partial E}{\partial a_k} = \sum_{i=1}^{M} 2 x_i^k (a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_N x_i^N - y_i) = 0 \quad \forall k = 0, \ldots, N$$

$\Rightarrow$ $(N+1)$ normal eqns. for $(N+1)$ unknown coeffs. a_i

$$a_0 M + a_1 \sum_{i=1}^{M} x_i + a_2 \sum_{i=1}^{M} x_i^2 + \ldots + a_N \sum_{i=1}^{M} x_i^N = \sum_{i=1}^{M} y_i$$

$$a_0 \sum_{i=1}^{M} x_i + a_1 \sum_{i=1}^{M} x_i^2 + a_2 \sum_{i=1}^{M} x_i^3 + \ldots + a_N \sum_{i=1}^{M} x_i^{N+1} = \sum_{i=1}^{M} x_i y_i$$

$\Big\}$ $\vdots$ etc.

Non-Polynomial General Least-Squares Fitting

Use a linearly independent set of functions $f_j(x)$ to fit data points (x_i, y_i) $i = 1, \ldots, M$

$$\tilde{F}(x) = a_0 f_0(x) + a_1 f_1(x) + \ldots + a_N f_N(x)$$

$$\Rightarrow F(a_0, \ldots, a_N) = \sum_{i=1}^{M} (\tilde{F}(x_i) - y_i)^2$$

$$= \sum_{i=1}^{M} \Big[\underbrace{\sum_{j=0}^{N} a_j f_j(x_i)}_{\tilde{F}(x_i)} - y_i \Big]^2$$

ECE 330

$$\frac{\partial F(a_0,\ldots,a_N)}{\partial a_k} = 0 \quad \forall k = 0,\ldots,N$$

$\Rightarrow$ System of normal equations:

$$\frac{\partial F(a_0,\ldots,a_N)}{\partial a_k} = 2\sum_{i=1}^{M}\sum_{j=0}^{N} a_j f_j(x_i) f_k(x_i) + \underbrace{[???]}_{\text{I didn't get this down}} = 0$$

$$\Rightarrow \sum_{i=1}^{M}\sum_{j=0}^{N} a_j f_j(x_i) f_k(x_i) = \sum_{i=1}^{M} y_i f_k(x_i)$$

Note: if data are not equally reliable/accurate, it might be desirable to weight the data (with more emphasis on accurate points):

$\Rightarrow$ minimize: $\tilde{F}(a_0,\ldots,a_N) = \sum_{i=1}^{M} w_i \left[\sum_{j=0}^{N} a_j f_j(x_i)\right]^2$

Optimization Schemes

(1) $f'(x) = 0$ or $\vec{\nabla} f(\vec{x}) = 0$

Root-finding methods

- Newton-Raphson (also for multi-dimensional systems) } uses $f'(x)$
  - "Newton" for 1D
  - "Newton-Raphson" for multi-dim
- Regular Falsi } use $f(x)$ with slope information
- Secant Method }
- Bisector Method } only $f(x)$

(2) Methods gradient descent or steepest descent

- Marquardt-Levenberg alg.

(3) Methods based on evolving rules and randomness

- Nelder-Mead alg.
  - aka Simplex alg.
  - aka Downhill-Simplex alg.
  - uses no derivatives; exclusively for multi-dimensional optimization

(4) (see next page)

↓

(4) Methods based on stochastic optimization
(use of random numbers)

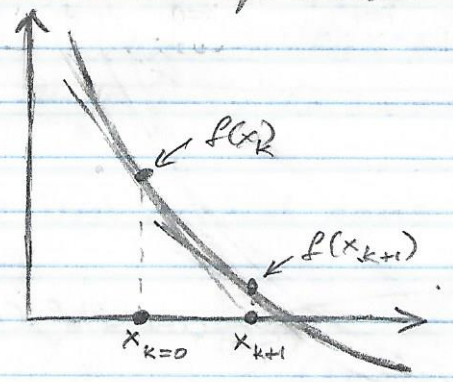
- Simulated Annealing (SA)
- Genetic Algorithms (GA)
- Evolutionary Algorithms (EA)
- Genetic Programming (GP)
  (John Koza, Stanford)

(Covered in this class)

⇒ Use no derivatives, only $f(x)$
$f(\vec{x})$

In particular, used for multi-dim. optimization

Newton-Raphson Method in 1D



Procedure:

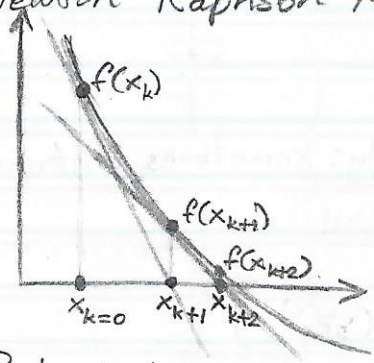
1) start with x-value, $x_{k=0}$
2) calculate $f(x_k)$
3) if $|f(x_k)| \leq$ user-defined $\varepsilon \Rightarrow$ root found $\rightarrow$ stop
   else: calculate $f'(x_k)$

[...] ← didn't get this down

ECE 330 10/18/2016

Root-finding methods

Newton-Raphson Method in 1D



Potential issues:

a.) $f(x)$ has to be differentiable (and ∴ also continuous)

b.) { method can run away ⇒ does not find the root
  { due to bad start value
  { method can get caught in endless loop
  → possible mitigation: choose a new start value

c.) $f'(x) = 0$ (e.g., asymptotic case)

d.) $f(x)$ may have more than one root → will catch
  { one of several roots but not necessarily all of them.
  → mitigation: choose different $x_{k=0}$ and hope to find several/all roots

---

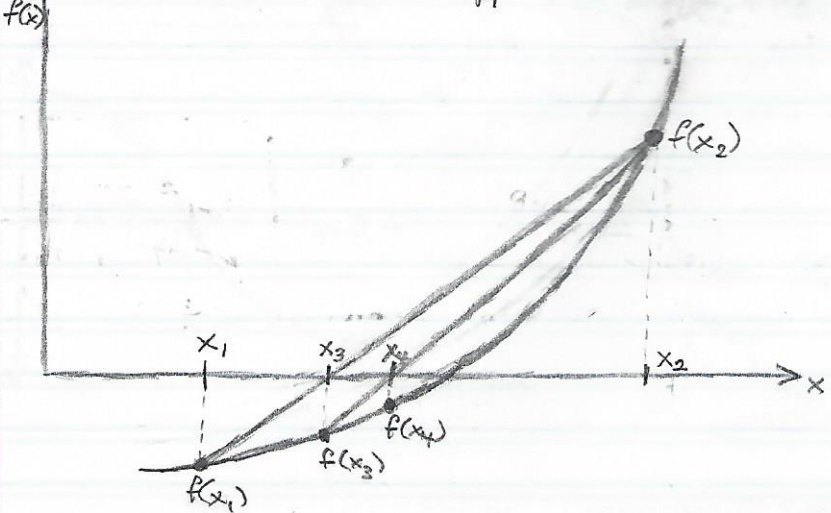
Newton-Raphson method converges quadratically near the root:
⇒ number of significant digits approximately doubles with each step.

Newton-Raphson is generalizable to multiple dimensions

False Position Method or "Regula Falsi"

uses only $f(x)$, NO first derivative $f'(x)$

brackets root within upper/lower bounds at all times



ECE 330

Note: initial values x_1 and x_2 must be chosen s.t. $f(x_1) \cdot f(x_2) < 0$

$$f(x) = f(x_k) + \underbrace{\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}}_{\text{slope}} (x - x_k)$$

Intersection of secant with x-axis:

$$f(x) = 0 \Rightarrow -f(x_k) \cdot \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} + x_k = x$$

$$\Rightarrow \boxed{x_{k+1} = \frac{f(x_k) \cdot x_{k-1} - f(x_{k-1}) \cdot x_k}{f(x_k) - f(x_{k-1})}}$$

Update formula for Regula Falsi

Important: if $(f(x_k) \cdot f(x_{k+1})) < 0$

$\Rightarrow$ new interval: $[x_{k+1}, x_k]$

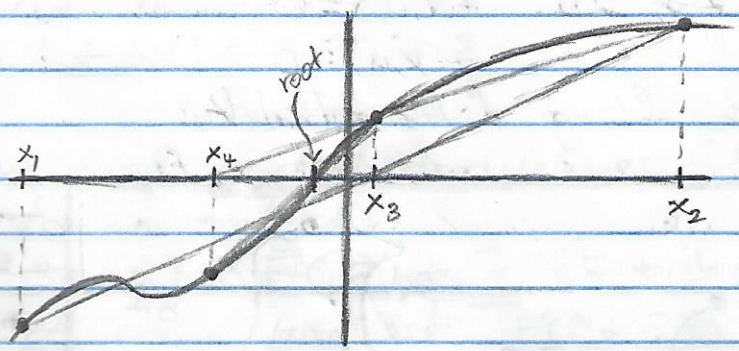
Always check if $|f(x_{k+1})| \leq \varepsilon$ before next iteration

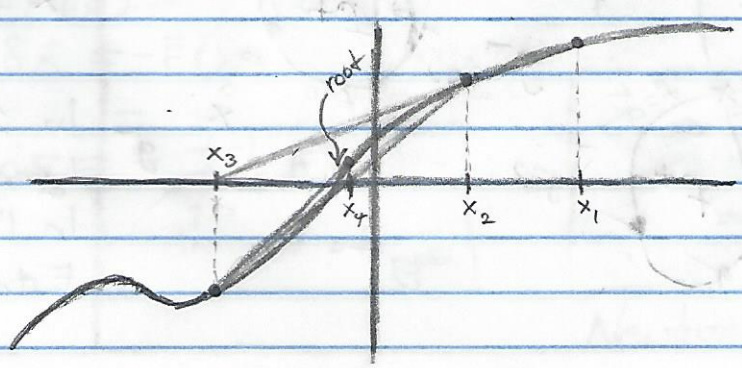
Secant Method:

uses only $f(x)$, NO first derivative $f'(x)$

does NOT have to bracket root!

$\Rightarrow$ location of start points x_1, x_2 arbitrary!





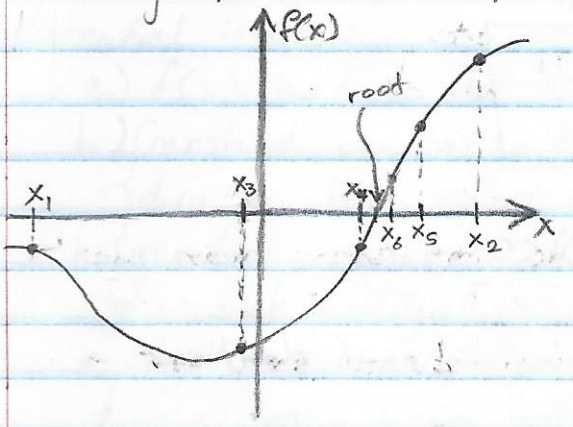
$$\boxed{x_{k+1} = \frac{f(x_k) x_{k-1} - f(x_{k-1}) x_k}{f(x_k) - f(x_{k-1})}}$$

Update formula for Secant Method

Important: Always use last two x values, regardless of whether $f(x_k)f(x_{k-1}) < 0$ or $f(x_k)f(x_{k-1}) > 0$

Bisection Method

- uses NO first derivative, only $f(x)$
- very robust
- always brackets root

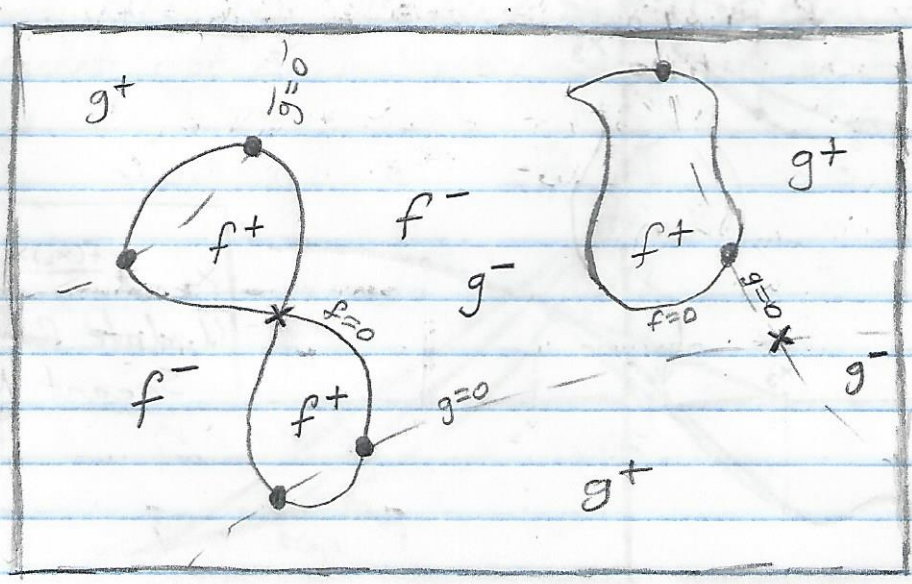


Pick midpoint x_3 in $[x_1, x_2]$
(i.e., bisect interval)
Check first if $|f(x_3)| \leq \varepsilon$
If $f(x_3)$ has the same sign as $f(x_1)$, use interval $[x_3, x_2]$
else $[x_1, x_3]$
Etc.

Multi-Dimensional Root Finding

Newton-Raphson Method for non-linear system of equations

Example for 2D: $f(x,y) = 0$
$g(x,y) = 0$
with f, g differentiable



— = f zero-contour
– – = g zero-contour
• = root of f, g
× = saddle point

In 2D, zero-contour hypersurfaces are lines
In 1D, points.
Etc.

ECE 330

Problems in higher dimensions N:

- find points that are zero in all N dimensions simultaneously
- $F_i(\vec{x}) = F_i(x_1, \ldots, x_N)$ generally unrelated
- zero-contours generally unrelated

$\Rightarrow$ Root-finding in N dimensions virtually impossible without further insight into $F_i(\vec{x})$ and additional information, such as how many roots to expect, whether roots are unique, etc.

$\Rightarrow$ Need excellent start points

Newton-Raphson Method for non-linear systems

have N func. relations to be zeroed w/ respect to x_i, $i = 0, \ldots, N-1$

$F_i(\underbrace{x_0, x_1, \ldots, x_{N-1}}_{\vec{x}}) = 0 \quad \forall i = 0, 1, \ldots, N-1$

$\Rightarrow \vec{F}(\vec{x}) = 0$

In neighborhood of $\vec{x}$ each function $F_i(\vec{x})$ can be expanded in Taylor series:

$$F_i(\vec{x} + \delta\vec{x}) = F_i(\vec{x}) + \sum_{j=0}$$

$\Rightarrow$ matrix of partial derivatives of functions F_i

$\Rightarrow$ Jacobian matrix

$$\vec{F}(\vec{x} + \delta\vec{x}) = \vec{F}(\vec{x}) + \underline{\underline{J}}\,\delta\vec{x} + \mathcal{O}(\delta\vec{x}^2)$$

Ignore error and set $\vec{F}(\vec{x} + \delta\vec{x})$ to 0

$$0 = \vec{F}(\vec{x}) + \underline{\underline{J}}\,\delta\vec{x}$$

$$\underline{\underline{J}}\,\delta\vec{x} = -\vec{F}(\vec{x})$$

$\Rightarrow$ $\boxed{\delta\vec{x} = -\underline{\underline{J}}^{-1}\vec{F}(\vec{x})}$ ($\underline{\underline{J}}^{-1}$: inverse of $\underline{\underline{J}}$)

$$\boxed{\vec{x}_{new} = \vec{x}_{old} + \delta\vec{x} = \vec{x}_{old} - \underline{\underline{J}}^{-1}\vec{F}(\vec{x}_{old})}$$

Newton-Raphson in N-Dimensions

ECE 330

Update eqns. → check $|F_i(\vec{x}_{new})| \leq \epsilon \quad \forall i = 0, \ldots, N-1$
if yes: done, i.e., have root
if not: next iteration

Linear Programming (not actually programming)

(for multi-dim. optimization
↳ "L.P."

LP is an N-dimensional optimization subject to linear constraints

Ex. 1: minimize $f(x_1, \ldots, x_N)$ $\quad x_i \in \mathbb{R}_0^+ \iff x_i \in \{x \in \mathbb{R} : x \geq 0\}$
subject to $x_i \geq 0 \quad \forall i = 1, \ldots, N$

⇒ example transformation of variables to accommodate constraints

$x_i = y_i^2 \quad \forall i = 1, \ldots, N$ with $y_i \in \mathbb{R}$
↳ (Could warp the probability distribution of values)

$x_i = |y_i|$ → (Does not warp the probability distribution of values)

⇒ minimize $f(y_1^2, y_2^2, \ldots, y_N^2)$ ← if $x_i = y_i^2$
or $f(|y_1|, |y_2|, \ldots, |y_N|)$ ← if $x_i = |y_i|$
no constraints in choice of y_i

Ex. 2: minimize $f(x_1, \ldots, x_N)$
subject to $l_i \leq x_i \leq u_i$ → or { with $u_i \geq l_i$

⇒ example transformation of variables
sinusoidal: $x_i = (u_i - l_i)\sin^2(y_i) + l_i$
↕ $\cos^2$
now, y_i is unconstrained.
alternative: $x_i = l_i + \underbrace{\text{ran3}()}_{\in [0,1)} \cdot (u_i - l_i)$

All of these examples are special cases of linear constraints of the following form:

$$\sum_{i=1}^{N} a_i x_i - b_j \geq 0$$

ECE 330

Linear Programming

Class of optimization problems where the objective function and all constraints are linear

$$f(\vec{x}) = \sum_{i=1}^{n} c_i x_i \text{ subject to: } \sum_{i=1}^{n} a_j x_i - b_j \geq 0 \quad \forall j = 1, \ldots, m$$

Example:

Transport company
2 types of trucks, A & B
A: cold capacity: $20\,m^3$
hot capacity: $40\,m^3$
B: cold capacity: $30\,m^3$
hot capacity: $30\,m^3$
Grocer needs trucks for $3000\,m^3$ cold stock and $4000\,m^3$ hot stock
A: \$30/km
B: \$40/km
Question: #A and #B to minimize total \$?

Procedure:

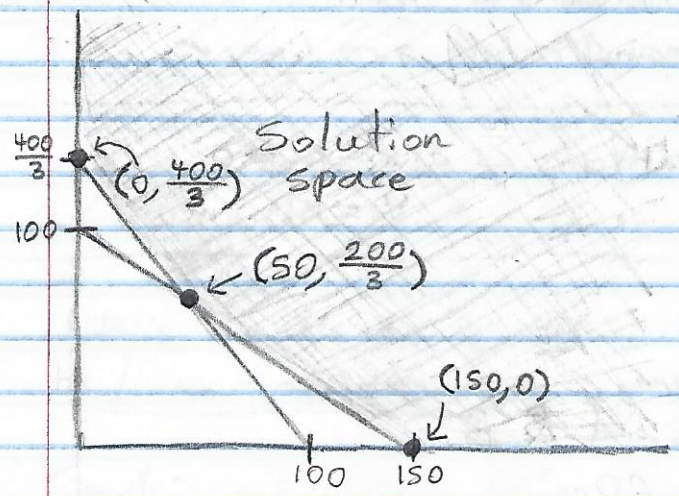
1) Choose unknowns
   x = # type A truck
   y = # type B truck
2) Write/formulate objective function
   $f(x,y) = 30x + 40y$
3) Define constraints as a system of inequalities
   Cold: $20x + 30y \geq 3000$
   Hot: $40x + 30y \geq 4000$
   } Explicit constraints: directly determined by task
   $x \geq 0$
   $y \geq 0$ and $x, y \in \mathbb{N}_0$ (natural numbers including 0)
   } Implicit constraints: implicitly determined by nature of problem

(See next page)

ECE 330

4) Find set of feasible solutions

$20x + 30y = 3000$
$40x + 30y = 4000$



5) Calculate coordinates of valid vertices

$(0, \frac{400}{3})$ invalid as-is → $(0, 134)$

$(150, 0)$ valid as-is → $(150, 0)$

$(50, \frac{200}{3})$ invalid as-is → $(50, 67)$

MISTAKE! ⇒ have to evaluate $(0,134)$, $(150,0)$, $(50,67)$ } valid vertex points

6) Evaluate objective function at each valid vertex

BASED ON MISTAKE!

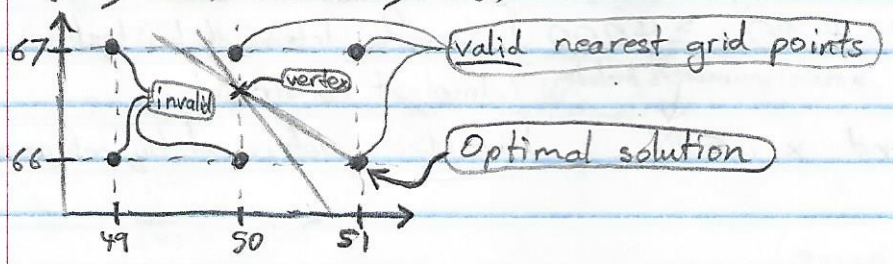
$f(0,134) = 5360$

$f(150,0) = 4500$

$f(50,67) = 4180$ ← minimal cost is 50 type A and 67 type B

7) Important: if LP problem requires any of the variables to be integers and you encounter fractions, then consider all valid integer values amongst nearest neighbors!

$(50, \frac{200}{3}) = (50, 66\frac{2}{3})$



ECE 330

Multi-dimensional optimization

So far: Newton-Raphson Method ← uses derivatives
Linear Programming ← uses function evaluations

Now: Levenberg-Marquardt Method ← uses derivatives
↳ refer to Numerical Recipes for description

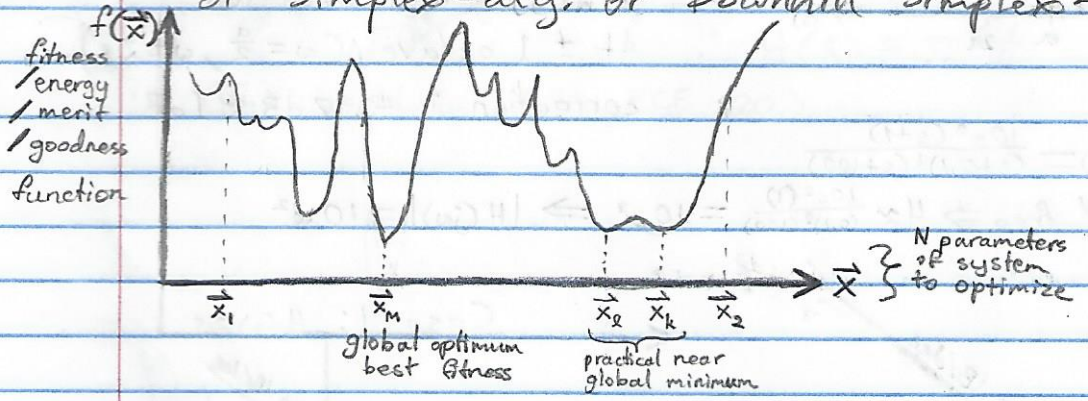
different names for the same algorithm:
- Nelder-Mead Method
- Simplex-Method
- Downhill Simplex Method

} uses function evaluations

Stochastic optimization methods:
- Simulated Annealing
- Genetic Algorithms
- Evolutionary Algorithms

} use function evaluations

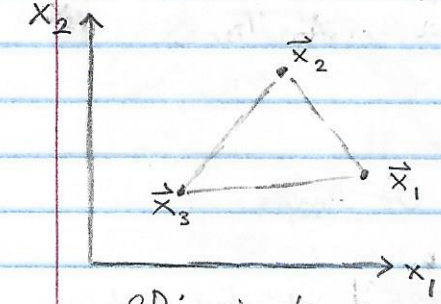
Nelder-Mead Multi-Dimensional Optimization
or Simplex-alg. or Downhill Simplex-alg.



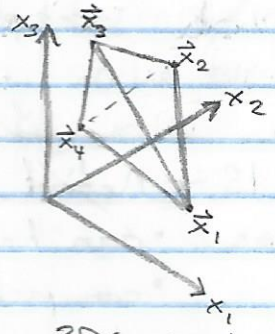
ECE 330

have N-dim. opt. problem

use at least (N+1) vertices for the simplex



2D: simplex is a triangle

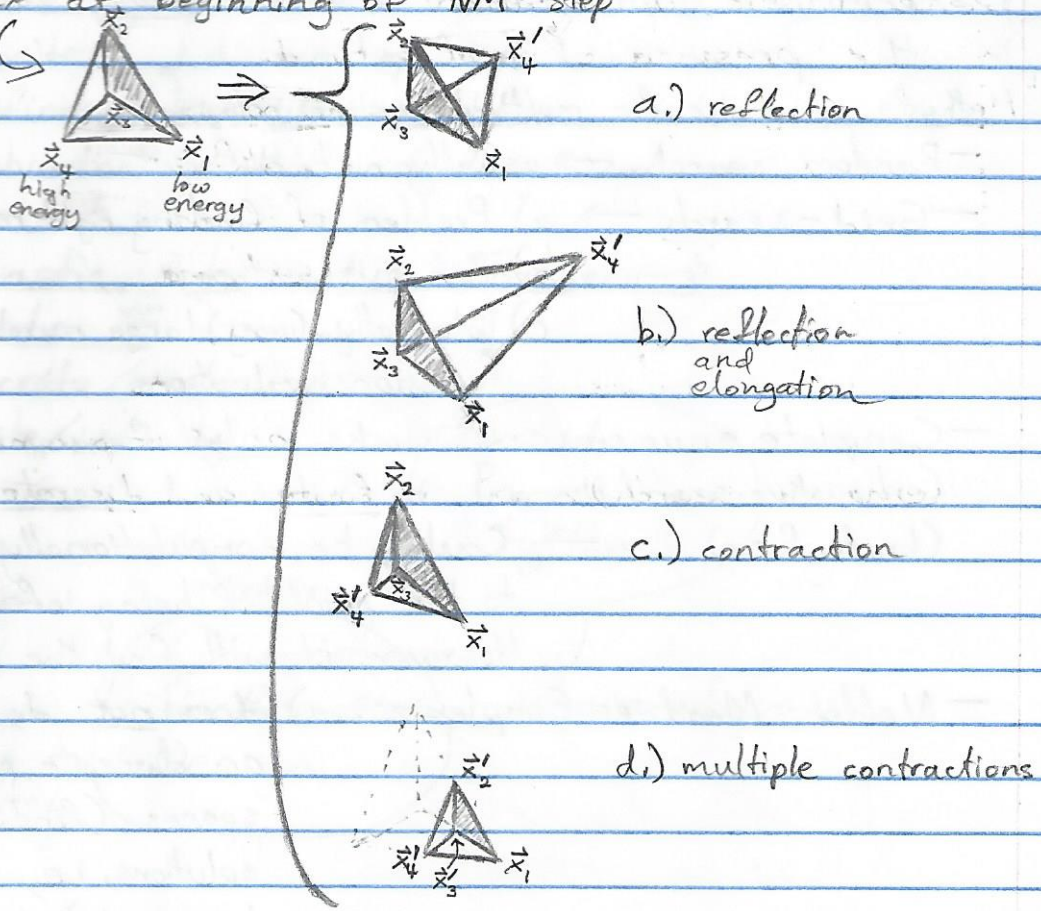


3D: simplex is a tetrahedron

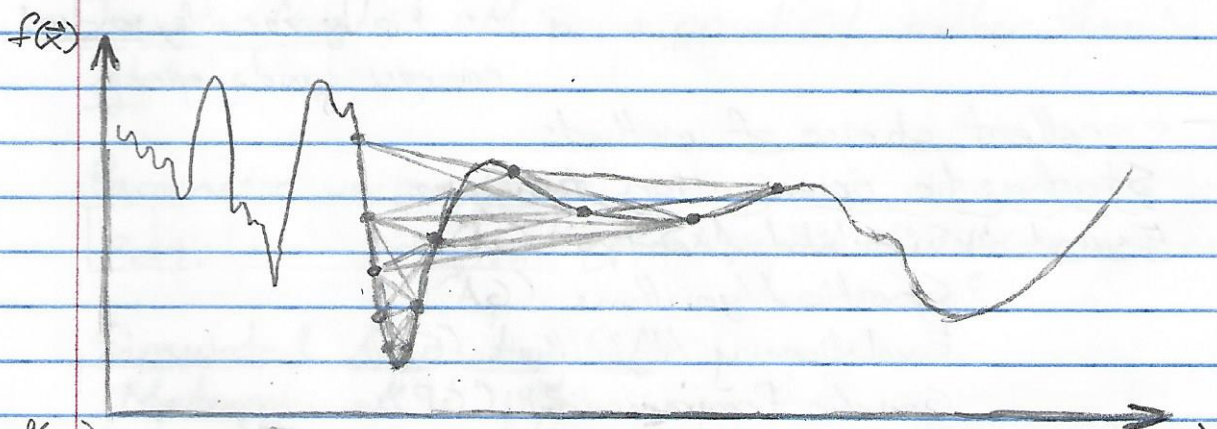
Simplex is essentially an N-dimensional polyhedron (with N+1 vertices)

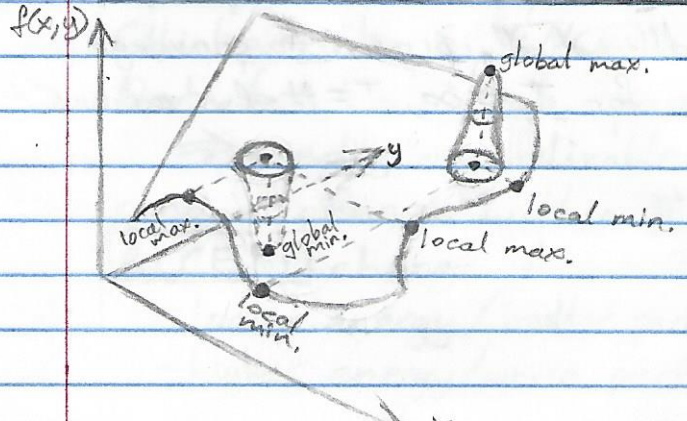
ECE 330

Simplex at beginning of NM-step



... etc.





ECE 330

Deterministic optimization algs. will NOT work (well) in the presence of local optima.

Methods of choice for multi-dimensional optimization

- Random search → usually not efficient
- Grid-search → a) Problem of finding right resolution
  b) can miss extrema
  c) Potentially (very) large number of function evaluations
- Complete enumeration: works only if parameter space is finite and discrete
  (exhaustive search)
  (brute force) → Could be computationally expensive to the point of being infeasible. However, it will find the global minimum
- Nelder-Mead or Simplex → a) does not do well on discrete parameter spaces (finding valid solutions, i.e., N-dim. hypercube)
  b) does not do too well on highly variable, non-local energy landscapes
- excellent choice of methods

Stochastic optimization techniques

Examples: Simulated Annealing (SA)
Genetic Algorithms (GA)
Evolutionary Algorithms (EA)
Genetic Programming (GP)

Note: Only SA is mathematically proven to converge to global optimum for $T \to \infty$, T = # of iterations

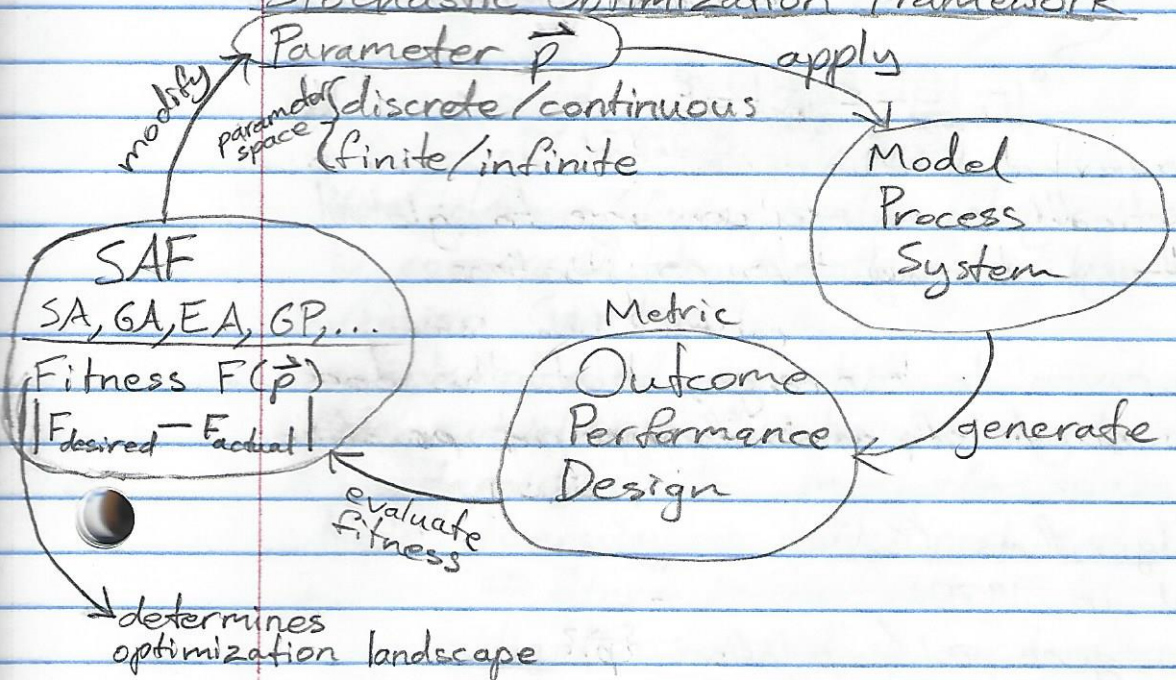
ECE 330

Types of search/parameter spaces

interval finite } { discrete values
interval infinite } { continuous values

all combinations possible, including mixed cases

Stochastic Optimization Framework



Task often to attain a practical, rather than global, optimum.

Parameter vector $\vec{p}$

| p_1 | p_2 | ..... | p_N |

$p_i \in$ {specific allowable or reasonable value range}

Simulated Annealing (SA)

Metropolis et al, 1953
(Kirkpatrick, SCIENCE)

Operates on single parameter vector $\vec{p}$
$\Rightarrow$ readily parallelizable

random change within $\vec{p}$ (i.e, point mutation)

ACCEPT change if:
- lower energy / better performance
- higher energy/worse performance only with a [...]

ECE 330

[↳ ...] certain Boltzmann-probability
REJECT change otherwise

Iterate until exit criterion is reached!
Exit criteria: - target fitness is achieved
- exceeded "allowable" number of iterations

$$|F_{desired} - F_{actual}| \leq \varepsilon_{user-defined}$$

SA mathematically proven to converge to global minimum, though it may take ∞ iterations

GA not proven to converge, but this does not make much of a difference in practice.

Genetic Algorithms (GA)

J. Holland in 1975.
operate on gene pool/population $\{\vec{p}_i\}$
→ population size = # of $\vec{p}$ in gene pool
(Note: parameter $\vec{p}$ is "genome")
⇒ GA is parallelizable, but not trivially due to "cross-talk", i.e., fitness exchange among CPUs.

1. $\{\vec{p}_i\}$ is modified through genetic operations:
   x%: point mutation (biologically motivated)
   y%: crossover (biologically motivated)
   z%: inversion (NOT biologically motivated)
2. Reevaluate fitness of modified gene pool
3. Create new population (generation) by adding offspring while keeping population size constant (i.e., we need to also remove constituents)
4. Iterate until exit criterion is reached!

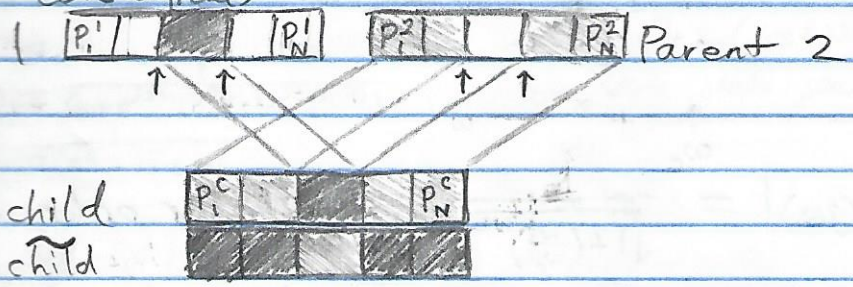
ECE 330

Genetic Operations in GAs

Point mutation (biological)

Pick a random parameter in $\vec{p}_i$ and modify it.

Crossover (biological)



Note: cleavage/clipping location chosen randomly, have to correspond at at both parent parameter vectors

Inversion (biological)

Invert order of parameters.

Note: not always feasible or easy to implement, depending on parameter spaces.

Note: diversity in gene pool can be lost
⇒ fitness becomes flat

mitigation: introduce random gene (new $\vec{p}$) on occasion (i.e., at a certain rate)

Note: Risk of replacing best gene with inferior offspring

mitigation: Elitist-approach: protects best performer, i.e, prevents it from being replaced
⇒ guarantees that you can maintain best solution

S.A. parameters

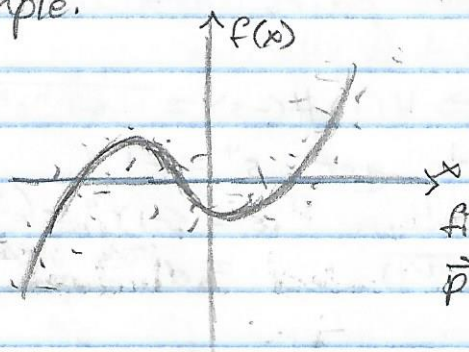
λ close to 0 → greedy downhill search

λ close to 1 → cool very slowly—accept anything for a long time.

$\text{rand}() < \underbrace{e^{\frac{-(E_{temp} - E_{best})}{T}}}_{\text{Boltzmann probability}}$

ECE 330

SA example:



fitting function: $f(x) = a_3x^3 + a_2x^2 + a_1x + a_0$

$\vec{p} = \langle a_0, a_1, a_2, a_3 \rangle$

Pick random values for $\vec{p}$

Set current and best fitness to (quadratic error of $f(\vec{p})$) $= F(\vec{p})$

Pick random or iterating parameter from $\vec{p}$ and modify it to get $\vec{p}'$

Set temp. fitness to (quadratic error of $f(\vec{p}')$) $= F(\vec{p}')$

If temp. fitness < best fitness, set best fitness to temp. fitness, set current $\vec{p}$ and $F(\vec{p})$ to $\vec{p}'$ and $F(\vec{p}')$, and restart loop

ECE 330

GA Pseudocode
- Point mutation
- Crossover
- Inversion

1.) Generate a population P of random genes

2.) Example of how to select individual gene for subsequent manipulation:

→ Weighted Roulette Wheel Selection

$sumFitness = \sum_{i=1}^{P} fitness(i)$

$rand = random_{[0,1)} * sumFitness$

```
i=0, partSum=0
do{
  i++
  partSum += fitness(i)
}while((partSum<rand) && (i<P))
                          (unnecessary check)
```

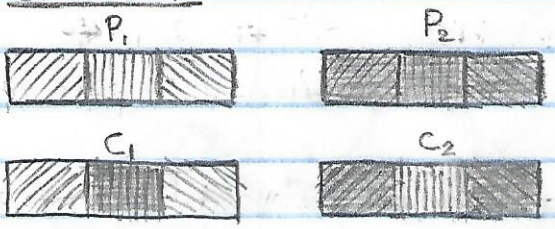
3.1) Inversion:

```
lchrom = length/dimension of a gene/parameter vector
for j=1,...,lchrom {
    gene_new[j] = gene_old[lchrom-j+1]
}
```

3.2) Point mutation:

```
for j=1,...,lchrom {
    gene_new[j] = mutation(gene_old[j], pMutation, &maxMutCntDown)
}
mutation(gene_old[j], pMutation, &maxMutCntDown){
    do{
        gene_new[j] = random value
    }while(gene_new[j] NOT valid OR gene_new[j]==gene_old[j])
    *maxMutCntDown--
}
```

3.3) Crossover:

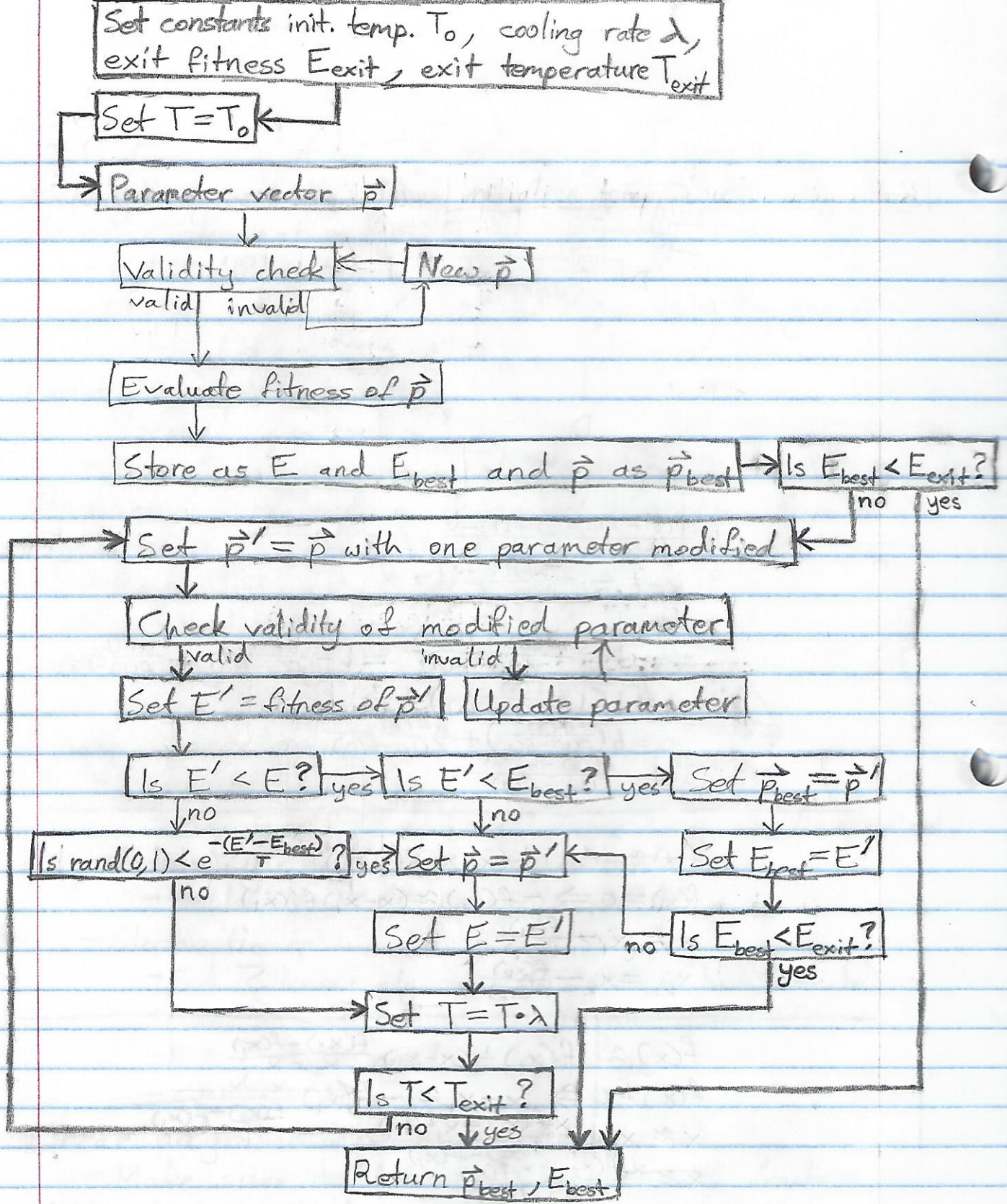


Often, crossover is combined with point mutations.

p_{cross} = P(crossover)
n_{cross} = counter for crossover
p_{mut} = P(mutation)
n_{mut} = counter for mutation
j_{cross} = crossover-point index
p1 = parent 1 } → { c1
p2 = parent 2 } → { c2

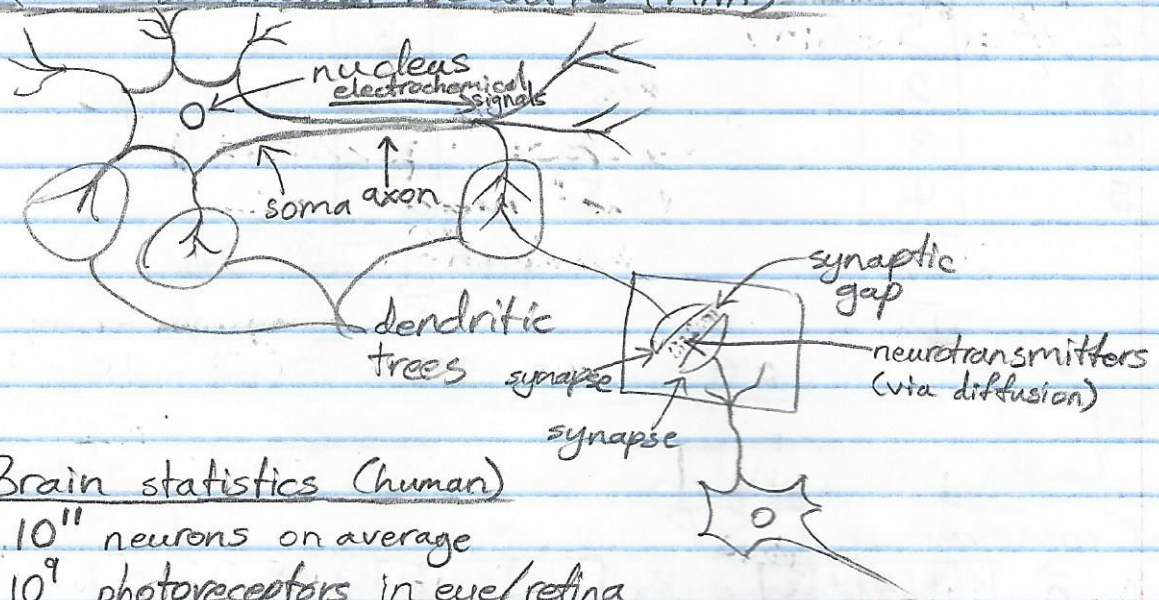
```
if random < pcross → perform crossover
{
    jcross = randint(1, lchrom)
    for j = jcross, ..., lchrom        } crossover and point mutation on part of gene
    {
        c1[j] = mutation(p2[j], pmut, maxMut)
        c2[j] = mutation(p1[j], pmut, maxMut)
    }
    for j = 1, ..., jcross - 1         } point mutation only on part of gene
    {
        c1[j] = mutation(p1, pmut, maxMut)
        c2[j] = mutation(p2, pmut, maxMut)
    }
}
```

ELSE perform point mutation only (on entire gene)


Set constants init. temp. T_0, cooling rate λ, exit fitness E_{exit}, exit temperature T_{exit}
Set $T = T_0$
Parameter vector $\vec{p}$
Validity check
New $\vec{p}$
valid
invalid
Evaluate fitness of $\vec{p}$
Store as E and E_{best} and $\vec{p}$ as $\vec{p}_{best}$
Is $E_{best} < E_{exit}$?
no
yes
Set $\vec{p}' = \vec{p}$ with one parameter modified
Check validity of modified parameter
valid
invalid
Set E' = fitness of $\vec{p}'$
Update parameter
Is $E' < E$?
yes
Is $E' < E_{best}$?
yes
Set $\vec{p}_{best} = \vec{p}'$
no
no
Is rand(0,1) $< e^{\frac{-(E'-E_{best})}{T}}$?
yes
Set $\vec{p} = \vec{p}'$
Set $E_{best} = E'$
no
Set $E = E'$
Is $E_{best} < E_{exit}$?
no
yes
Set $T = T \cdot \lambda$
Is $T < T_{exit}$?
no
yes
Return $\vec{p}_{best}$, E_{best}

ECE 330

Artificial Neural Networks (ANN)



Brain statistics (human)

- 10^{11} neurons on average
- 10^{9} photoreceptors in eye/retina
- 10^{6} fibers in optic nerve (intelligent 1000:1 downsampling)
- 10^{4} couplings per neuron
- 10^{-2}-second response time (10 ms)
- 10^{15} overall neural couplings

Biology

On a basic level, the soma sums up inputs via dendritic trees (from other neurons)

$\therefore \Rightarrow$ local field

threshold not exceeded → neuron stays dormant (quiescent)

threshold exceeded → neuron fires

$\Rightarrow$ generates electrochemical signals

$\Rightarrow$ propagate along axon to other neurons

neuron fires/spikes $\Rightarrow$ generation of a spike train

$\Rightarrow$ information encoded in firing rate

$\Rightarrow$ signal propagates via axon

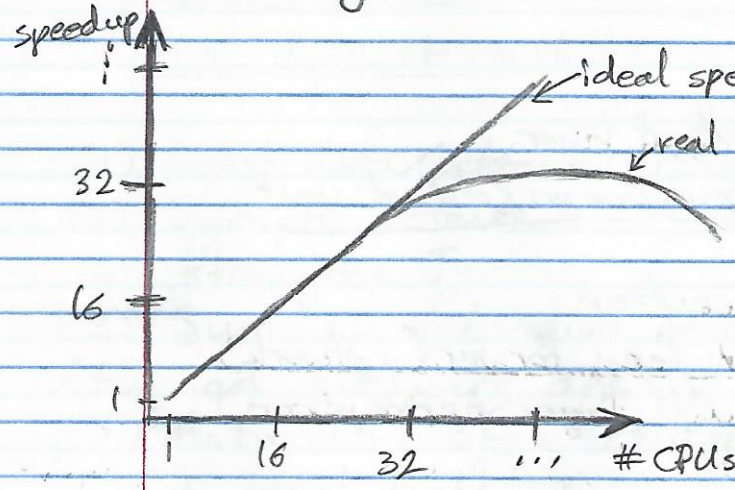
$\Rightarrow$ provides input to other dendritic trees via dendritic trees

ECE 330

Capability of brain:

a.) # of neurons
b.) # of interconnections (synaptic couplings)
c.) architecture (i.e., spacial organization)
d.) strengths of individual interconnections
(akin to a data-bus bandwidth)
→ "Bonhoeffer effect"

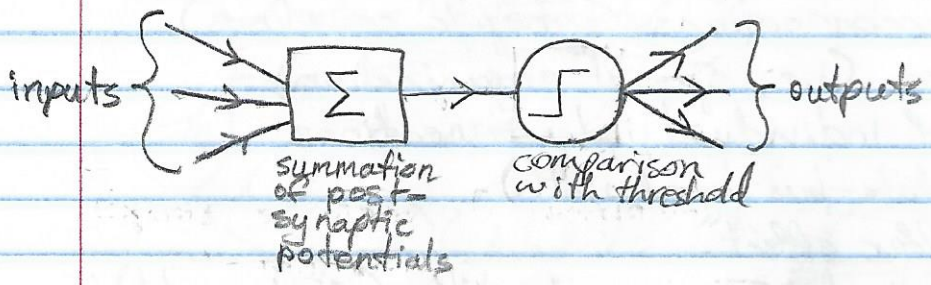
in-vivo (biological):	in-silico (CPU-world):
single neuron: slow simple	single CPU: fast complex
many neurons: fast complex (brain)	many CPUs: "slow down" "simple" (more of the same) but no surprises
fault-tolerant	fault-intolerant
robust	not robust
low power consumption	high power consumption
⇒ emergent behavior	⇒ non-emergent behavior (predictive behavior)



Factors affecting speedup:
- task parallelizable?
- cross-talk/data dependency
- job/task distribution
- architecture (tree, ring, torus)
- local/global shared RAM/HDD
- databus bandwidth

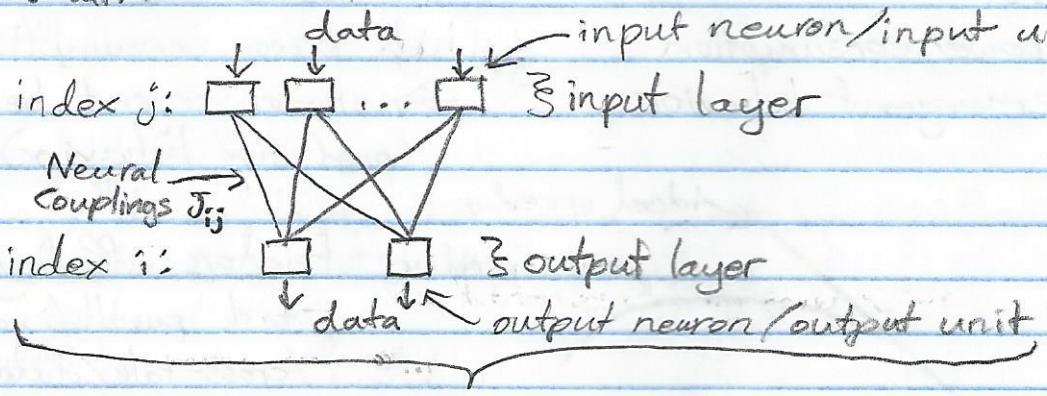
ECE 330

1943, McCulloch & Pitts
→ McCulloch-Pitts neuron (MP neuron)



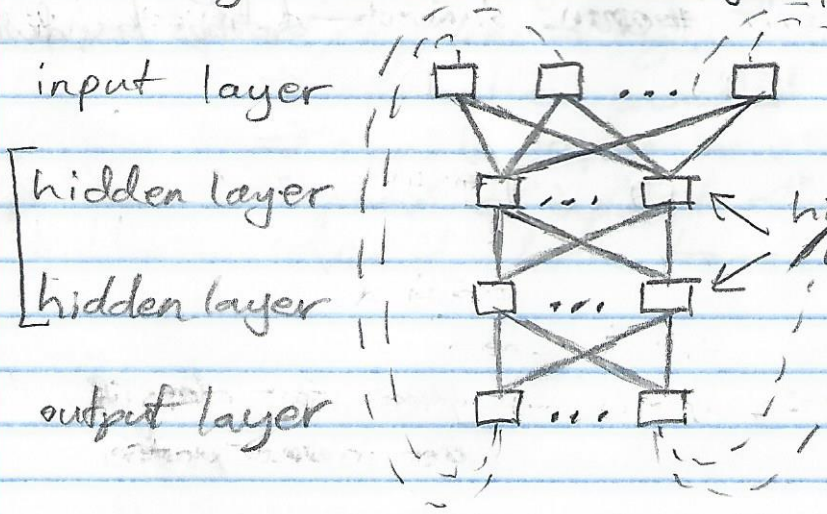
Classes of ANN:

(single / multi)-layer feed forward ANNs
also known as
(single / multi)-layered perceptrons



simple perceptron
(single-layer feed forward network)

Multi-layer FFANN (Multi-layer perceptron)

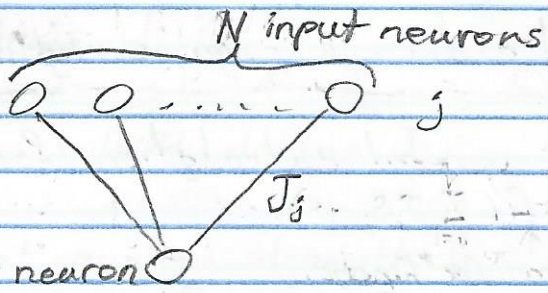


→ recurrent network:
uses (part of) output as feedback for next iteration
→ dynamic network

ECE 330

Simple perceptron

input layer



$\Rightarrow$ all couplings J_j form a coupling vector $\vec{J}$ (N-dimensional)

$J_j \in \mathbb{R}$: simple spherical perceptron with $\sum_{j=1}^{N} J_j^2 = N$

$\Rightarrow$ coupling vector $\vec{J}$ on N-dimensional sphere of radius $\sqrt{N}$

$J_j \in \{-1, 1\}$

binary perceptron

aka

simple Ising perceptron

How does it work?

(1) Input data/pattern presented as synaptic potentials S_j
(2) Synaptic potentials S_j multiplied by neural coupling strengths J_j
(3) Sum up individual results as <u>local field</u>
(4) and compare against a given threshold
(5) Activation of a neuron depends on magnitude of local field

All of the above as a formula:

$$\text{output } O = \operatorname{sgn}\left(\sum_{j=1}^{N} J_j S_j - \delta\right)$$

sgn: signum; δ: threshold

with $\operatorname{sgn}(x) = \begin{cases} +1, & x > 0 \\ -1, & \text{else} \end{cases}$

$\Rightarrow O \in \{-1, +1\}$ $\Rightarrow$ used for classification of input patterns

ECE 330

Final exam will cover everything except ANNs, Matlab, and Gnuplot.

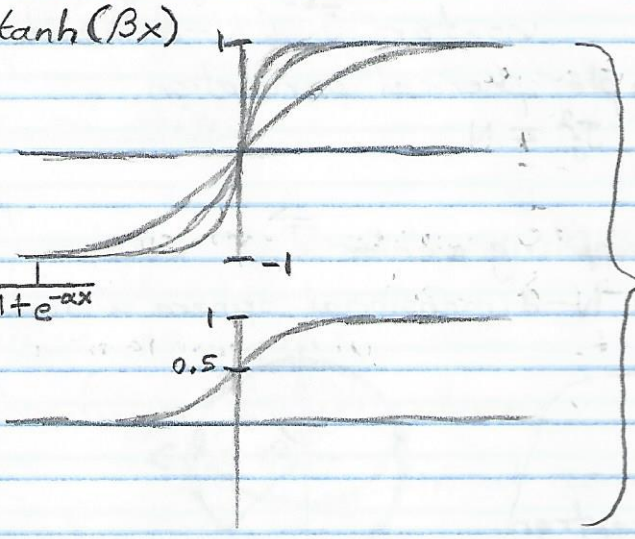
More general calculation for output:

$$O = f\left(\sum_{j=1}^{N} J_j S_j - \delta\right)$$

f: transfer function

Examples of transfer function:

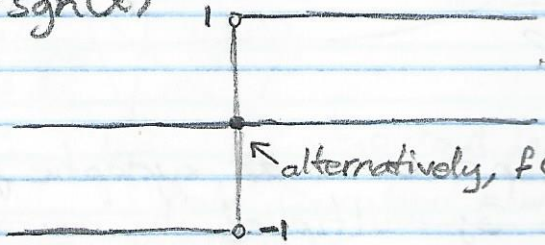
$f(x) = \tanh(\beta x)$



$f(x) = \frac{1}{1+e^{-ax}}$

} sigmoidal functions

$f(x) = \text{sgn}(x)$



↖ alternatively, $f(0)$ could be -1 (or 1).

Perceptrons used for:

(I) pattern storage/recognition, data classification
(II) generalization/rule learning
(III) control of dynamic systems

} can be treated analytically with statistical mechanics methods, usually in thermodynamic limit ($N \to \infty$) to calculate storage capacity and generalization error.

What makes up a neural network?

given: # of input neurons
# of output neurons

TBD: (ANN architecture (empirical))
- # of hidden layers
- # of neurons in each layer
- sets of couplings between layers
- strength of each coupling

ECE 330

How to get the couplings J_{ij}?

(1) learning from training examples: {inputs, associated correct outputs}

(2) a priori design, i.e., explicitly define J_{ij}

How to determine neural coupling strengths?

simple perceptron:

state of output neurons:

$$S_i = f\left(\underbrace{\sum_k J_{ik}\sigma_k}_{\text{local field}} - \underbrace{\nu^e}_{\text{threshold}}\right)$$

often $\nu^e = 0$ without restriction

$$\Rightarrow S_i = f\left(\sum_k J_{ik}\sigma_k\right)$$

So how to choose J_{ik} such that certain $\vec{\sigma}$ leads to desired output $\vec{S} = \vec{\xi}$

and this for a number of cases μ $\forall \mu = 1, \ldots, p$ (p = # of patterns)

so, $S_i^\mu = S_i(\sigma^\mu) = \xi_i^\mu$

σ_k^μ: pattern μ @ input neuron k

ξ_i^μ: desired output at output neuron i for pattern μ

S_i: actual output at output neuron i for pattern μ

Explicit formula for J_{ik} from σ_k^μ and ξ_i^μ not known!

$\Rightarrow$ Construct iterative procedure which converges to desired values of J_{ik}, if those values exist (not always possible).

Consider $f(x) = \text{sgn}(x) \rightarrow S_i, \xi_i \in \{-1, 1\}$

for simple pattern we have

$S_i = \xi_i$ or $S_i = -\xi_i$ } adjust if $S_i \neq \xi_i$

Adjustment of couplings:

deviation $\delta J_{ik} := J_{ik}^{new} - J_{ik}^{old}$

$$\delta J_{ik} = \varepsilon(\xi_i\sigma_k - S_i\sigma_k)$$

ε: learning rate, by choice

ECE 330

$\Rightarrow$ Update to J_{ik}

$$J_{ik}^{new} = J_{ik}^{old} + \varepsilon(\xi_i^{\circ}\sigma_k - s_i\sigma_k)$$

$\Rightarrow$ Perceptron Learning Rule

$\forall$ patterns μ: $1, \ldots, p$

$$\delta J_{ik} = \varepsilon \sum_{\mu=1}^{p} (\xi_i^{\mu} - s_i^{\mu})\sigma_k^{\mu}$$

$$\Rightarrow J_{ik}^{new} = J_{ik}^{old} + \varepsilon \sum_{\mu=1}^{p} (\xi_i^{\mu} - s_i^{\mu})\sigma_k^{\mu}$$

For multi-layer perceptrons
Error-Backpropagation Alg.
by Rumelhart et al.

ECE 330 — Midterm One Topics

No MATLAB
No GnuPlot
Distinction between Numerical Recipes & GNU Sci. Lib.
2pt impr. 2pt 3pt
☑☑ full derivation
leading error terms
Go over homework
Num. Integ.
☑☑ Trap. rule & deriv
☑☑ Simp. rule & deriv
☑☑ Single/multiple intervals
GLI full end result
Rodriguez formula: $L_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2-1)^n$
Specifics of Leg. polys.
Orthogonal
Normalization factor
Expansion of general poly. to Leg. polys.
Roots in (-1, 1)
Cancellation/Dirac notation
Exact up to $2n-1$ for polynomials
2^{nd}-order D.E. → coupled 1^{st}-order D.E.s
☑☑☑ All D.E. methods (EM, IEM, RK4)
☑☑ Derivations of EM, IEM
☑☑☑ Graphical depictions (rough sketches)
Order of error (h^2, h^3, h^5)
Everything about electric lens
For sure, Laplace eqn. in Cartesian/cylindrical coords.
Reduced Laplace eqn.
Method of Successive Over-Relaxation
Not update eqns. for grid pts.
Not z-axis update eqn.
Do know general update eqn. involving ϕ, U
Not Gauss Elimination (until next midterm)

$$\langle L_m | L_n \rangle = \delta_{mn} \cdot \frac{2}{2n+1}$$
$$= \begin{cases} 0 & \text{for } m \neq n \\ \frac{2}{2n+1} & \text{for } m = n \end{cases}$$

ECE 330

Know Rodriguez formula

Know end result of of GLI formula

$$\int_a^b f(u)\,du \approx \frac{b-a}{2}\sum_{i=1}^{n} w_i\, f\left(\frac{(b-a)x_i}{2}+\frac{b+a}{2}\right)$$

↑ replace with "=" for polynomials of degree $\leq 2n-1$ (or $< 2n$)

Expanding polynomials into Legendre polynomials:

$$p(x) = x^2+2 = \sum_{i=0}^{2} c_i L_i(x)$$

$L_0(x) = 1$

$L_1(x) = x$

$L_2(x) = \frac{3}{2}x^2 - \frac{1}{2}$

$$\frac{x^2+2}{\frac{3}{2}x^2-\frac{1}{2}}$$

$c_2 = \frac{2}{3} \Rightarrow p(x) - c_2 L_2 = x^2+2-x^2+\frac{1}{3}$

$= \frac{7}{3}$

$\frac{7}{3}$ does not contain $x \Rightarrow c_1 = 0$

$c_0 = \frac{7}{3} \Rightarrow \frac{7}{3} - c_0 L_0 = 0$

$p(x) = x^2+2x$

$c_2 = \frac{2}{3} \Rightarrow p(x) - c_2 L_2 = \cancel{x^2}+2x-\cancel{x^2}+\frac{1}{3}$

$= 2x+\frac{1}{3}$

$c_1 = 2 \Rightarrow 2x+\frac{1}{3} - 2L_1 = \cancel{2x}+\frac{1}{3}-\cancel{2x}$

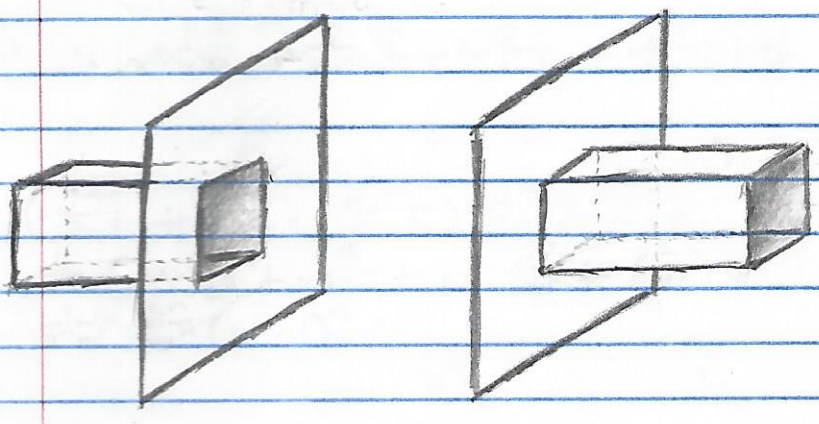
$= \frac{1}{3}$

$c_0 = \frac{1}{3}$

ECE 330

$p(x) = x^2 + 2x$

$c_2 = \frac{2}{3} \Rightarrow p(x) - c_2 L_2 = x^2 + 2x$



Memorize Laplace operator in cartesian coords. and definitely in cylindrical coords.

Read questions carefully. There may be trick questions.

Go through example test questions!

ECE 330

$$\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)\phi=0$$

Rectangular Laplace eqn.

$$\left(\frac{\partial^2}{\partial z^2}+\frac{1}{r}\frac{\partial}{\partial r}+\frac{\partial^2}{\partial r^2}+\frac{1}{r^2}\frac{\partial^2}{\partial \varphi^2}\right)\phi=0$$

$$\frac{1}{r^2}\left(r^2\frac{\partial^2}{\partial z^2}+r\frac{\partial}{\partial r}+r^2\frac{\partial^2}{\partial r^2}+\frac{\partial^2}{\partial \varphi^2}\right)\phi$$

Cylindrical Laplace eqn.

Rect $\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)\phi$

Polar $\frac{1}{r^2}\left(r^2\frac{\partial^2}{\partial z^2}+r\frac{\partial}{\partial r}+r^2\frac{\partial^2}{\partial r^2}+\frac{\partial^2}{\partial \varphi^2}\right)\phi$

Rodriguez $\frac{1}{2^n n!}\frac{\partial^n}{\partial x^n}(x^2-1)^n$

ECE 330 – Exam 2 Topics

Everything strictly after electric lens

Gaussian elimination & GJE
- Equation systems
- Matrix inversion (could be impossible – fair game)
  - Make sure $\underline{\underline{A}} \cdot \underline{\underline{A}}^{-1} = \underline{\underline{I}}$

Least-squares regression
- Normal equations
- Transforming exponentials

Stochastic optimization framework (diagram from class)

Optimization methods
- Root-finding methods
  - Newton-Raphson
  - Regula Falsi
  - Secant
  - Bisection

  } Know derivations and pitfalls
- Steepest descent
  - Marquardt-Ledenberg (name only, spelling lenient)
- Evolving rules and randomness
  - Nelder-Mead/Simplex/Downhill Simplex
- Stochastic optimization
  - SA } know algorithms (down to pseudocode)
  - GA }
  - EA } don't need to know
  - GP }

Linear programming (definitely on test)
- Make sure to know how to deal with fractions

Multi-dim. methods
- Random search
- Grid search
- Nelder-Mead (looks like an octopus/amoeba)
- Exhaustive search

Know everything about SA algorithm

Know three GA operators

ECE 330

Final Exam

Everything listed for exams 1 & 2, with the same specified levels of detail. Some material might not have been on a prior exam, but it will all be from the lists of viable topics. Everything after exam 2 was on neural networks, so it won't be on the final.

Linear programming likely
EM/IEM/RK4 likely (& transforming 2nd-order to coupled 1st-order)
GLI likely
Two-point/Improved two-point/Three-point likely
"Electric lens is also part of it"
- Know Laplace eqn. in rect/cyl. coordinate systems
- Don't need update eqns. per se
- Know Laplace operator

Root-finding methods
- Derivation of update eqn, involving inverse Jacobian matrix

Homework, excluding programming
S.O.F.
S.A.
- How it can turn into greedy downhill search
- How it can turn into random search

G.A.
- Operators, biological motivation, etc.

- ☐ Least-squares regression (normal equations)
- ☐ S.O.F.
- ☐ Linear programming
- ☐☐☐ 2-pt./derivation/error
- ☐☐☐ Impr. 2-pt./derivation/error
- ☐☐☐ 3-pt./derivation/error
- ☐☐ T.R./derivation
- ☐☐ S.R./derivation

G.L.I.

- ☐ Rodriguez formula: $L_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2-1)^n$

Orthogonality } $\langle L_m | L_n \rangle = \delta_{mn} \cdot \frac{2}{2n+1}$

Normalization factor } $= \begin{cases} 0 & \text{for } m \neq n \\ \frac{2}{2n+1} & \text{for } m = n \end{cases}$