Content

**Sigma Algebra**

**Probability Measure and Change of Measure**

**Conditional Expectation**

**Basic Concept of Statistics**

**Random Walk**

**Ito Integral**

**Ito formula and Geometric Brownian motion**

1. **Sigma algebra, random variable, filtration**

**1.1 Sigma algebra**

1. **Sigma algebra definition**
2. Include empty set and whole set
3. Include the complement of any element itself
4. Closed under countable union
5. **Sigma algebra example**

Tossing 0 time

Check 1)

*F*\_0 = { 0,}

Tossing once

Check 1 \Omega = {H, T}

Check 2 H\_c = T, T\_c = H

Check 3 H U T = \Omega

So

F\_1 = {0, \Omega, H, T}

Tossing twice

Check 1 \Omega = {HH, HT, TH, TT}

Check 2 HH\_c, HT\_c, TH\_c, TT\_c

Check 3 HH U HT, HH U TH, HH U TT, HT U TH, HT U TT, TH U TT

HH U HT U TH = TT\_c,

HH U HT U TT = TH\_c,

HH U TH U TT = HT\_c,

HT U TH U TT = HH\_c

So

F\_2 = { 0, \Omega, HH, HT, TH, TT,

HH\_c, HT\_c, TH\_c, TT\_c

HH U HT, HH U TH, HH U TT, HT U TH, HT U TT, TH U TT,

TT\_c, TH\_c, HT\_c, HH\_c}

1. **Why define sigma algebra?**

On top of the sigma algebra, we can define the probability, because the object that probability measure takes is the sigma algebra.

* 1. **Filtration**

Consider a sequence of coin toss

For the first toss, we get *F*\_1

For the first and second toss, we get *F*\_2

For the first n tosses, we get *F*\_n

The collection of sigma algebra *F*\_1, *F*\_2… *F*\_n is called a **Filtration**.

* 1. **Random variable**

A random variable is function from \Omega -> R, which satisfies for all of the subsets {\omega in \Omega, X in Borel} is in \sigma-algebra *F*

* 1. **Sigma algebra generated by a random variable and measurable function**

Give consider a random variable S: \Omega -> R, for every open set in R, the collection of their inverse image forms an sigma algebra, and it is called the **sigma algebra generated by S. And S is called F-measurable**.

1. **Probability measure and examples**

Give a random variable X(w)=w where w in [0,1]

Define probability measure

P(a<x<b) = b-a ,

So this is a uniform measure.

Define another probability measure

P’(a<x<b) = b^2 – a^2

So this is non uniform measure

1. To justify they both are probability measure

Check P[0,1] = 1; P(0)=0;

P’[0,1] =1; P’(0)=0;

1. **Change of measure**

Consider the transformation

P’(w) = \int \_a ^b 2w dw = \int \_a ^b 2w dw = \int\_a ^b 2w dP(w)

dP’(w) = Z(w) dP(w) where Z(w)= 2w

**This is the change of measure**.

1. **Change of measure in normal distribution**

If X is N(0,1), let Y = X+ u, so Y is N(u,1)

Define Z(w) = exp(-uX – ½ u^2)

We are able to show two things

1 Z > 0

2 EZ =1 i.e. \int Z(w)dP(w) = 1

Because EZ = \int exp(-ux -1/2u^2) \frac{1}{\sqrt{2pi}} exp(-1/2x^2) dx

= \frac{1}{\sqrt{2pi}} \int exp(-1/2(x+u)^2) dx

= \frac{1}{\sqrt{2pi}} \int exp(-1/2(y)^2) dy

=1

**So P’(w) = \int Z(w) dP(w) is a new Prob measure**

The pdf of Y under the new measure is

P’(Y<=b)

= \int\_{Y(w)<=b} dP’(w)

= \int {Y(w)<=b} Z(w) dP(w)

= \int 1\_{X(w)<=b-u} exp(-uX- ½ u^2 )dP(w)

=\int 1\_{X(w)<=b-u} exp(-uX- ½ u^2 ) cdf(N(0,1)) dx

=\frac{1}{\sqrt(2\*pi)} \int ^{-\inf}^{b-u} exp(-ux – 1/2u^2 -1/2x^2) dx

=\frac{1}{\sqrt(2\*pi)} \int ^{-\inf}^{b-u} exp(-1/2 (x+u)^2) dx

(change x back to y)

=\frac{1}{\sqrt(2\*pi)} \int ^{-\inf}^{b} exp(-1/2 (y)^2) dy

**= cdf of N(0,1)**

1. **Conditional expectation**
2. **Def**
3. E[X| *G* ] is *G* measurable
4. \int\_A E[X|G](w)dP(w)= \int\_A X(w)dP(w**)**
5. **Example**

Consider 3 toss case, H with prob p, T with prob q

Def random variable S

S\_0(w) = 4 for all w

S\_{n+1}(w) = 2S\_{n}(w) if w\_{n+1}= H

½ S\_{n}(w) if w\_{n+1} = T

Expectation of 3 tosses random variable S\_3 give the first two is HH

E\_2(S\_3 | HH) = p S\_3(HHH) + qS\_3(HHT)

E\_2(S\_3 | HT) = p S\_3(HTH) + qS\_3(HTT)

E\_2(S\_3 | TH) = p S\_3(THH) + qS\_3(THT)

E\_2(S\_3 | TT) = p S\_3(TTH) + qS\_3(TTT)

E\_2(S\_3 | HH) P(HH) = prob(HHH) S\_3(HHH) + prob(HHT) S\_3(HHT)

E\_2(S\_3 | HT) P(HT)= prob(HTH) S\_3(HTH) + prob(HTT) S\_3(HTT)

E\_2(S\_3 | TH) P(TH)= prob(THH) S\_3(THH) + prob(THT )S\_3(THT)

E\_2(S\_3 | TT) P(TT)= prob(HTH) S\_3(TTH) + prob(TTT) S\_3(TTT)

This confirms def 2), for A = HH or HT or TH or TT

\int \E\_2(S\_3|*G*)(w) dP(w) = \int \_A X(w)dP(w)

1. **Properties**
2. If X is *G* measurable E[XY|*G*] = X E[Y|*G*]
3. If *G* is a subset of *H*

E[E[X|*G*|*H*]] = E[X|*H*]

1. If X is independent of *G*

E[X|*G*] = EX

1. **Basic Concept of Statistics**

**4.1 Independent vs uncorrelated**

**a. Independent**: Joint probability equals the product of individual probability

**b. Uncorrelated:** Covariance is zero**.**

**c. Relationship:**

Independence implies uncorrelated while uncorrelated does not imply independence.

**Example:** Normal random variable with zero mean. Y = X^2

Cov(X, Y) = E(XY) – EX EY

=E(XY) = E(X^3) = 0

However X and Y are clearly dependent.

* 1. **Law of large number**

1. **Weak law of large number**

lim^{n -> \inf} Pr(| \bar X –u | > \epsilon ) =0

1. **Strong law of large number**

Pr (lim^{n -> \inf} \bar X = u ) =1

1. **Difference**

In weak case, | X – u | > \epsilon can happen infinite times, however in the strong case, it does not. An example would be a series that is conditionally convergent.

* 1. **Central limit theorem**

Y\_1, Y\_2, …Y\_n iid, E(Y) = u, Var(Y) = \sigma^2,

/sqrt{n}(\frac{Y\_1+…+Y\_n}{n} - u) converge to distribution N(0, \sigma^2)

**4.4 Confidence Interval**

1. **Large sample n >30**

Z = \frac{\hat u - u}{ S/\sqrt{n}} is N(0, 1)

Given a \alpha and z\_{\alpha}

which satisfies P(|z|>z\_{1/2 \alpha}) = \alpha

\alpha is the tail area of the pdf.

Then P(-z\_{1/2 \ahlpa} <= \frac{\hatu – u }{\sigma}) = 1 - \alpha

Solve for u = [ u-, u+]

We interpret the probability that \hat u falls into [u-, u+] is 1-\alpha

1. **Small sample n<30**

**Principles are the same, except the distribution we use is t distribution with n-1 freedom**

t = \frac{\hat u - u}{S/\sqrt{n}}

1. **Random walk**

**5.1 Symmetric Random Walk**

**a. def**

Starting from coin toss, def:

X\_j = 1 if w\_j = H

= -1 if w\_j = T

Let M\_k = \sum\_{j=1} ^{k} X\_j, M\_k is **symmetric random walk, and its distribution is binomial**

**b. Expectation and Variance**

E(X\_j) =0, Var(X\_j) =1,

Var(M\_k – M\_l) = \sum\_{i=l}^{k} Var(X\_i) = k – l;

**b. Martingale Property**

E(M\_l| F\_k)

=E((M\_l – M\_k + M\_k) | F\_k)

=E((M-l – M\_k) | F\_k) + E(M\_l | F\_k)

= 0 + M\_k = M\_k

* 1. **Scaled Random Walk**

1. **Scaled symmetric random walk Definition**

W^{(n)}(t) = 1/\sqrt{n} M\_{nt}

= 1/\sqrt{n} \sum\_{i=1} ^{nt} X\_^{(n)}(t)

= \sum\_{i=1} ^{nt} 1/\sqrt{n} X\_^{(n)}(t)

1. **Expectation and Variance**

E(W^{(n)}(t) – W^{(n)}(s)) =0,

Var(W^{(n)}(t) – W^{(n)}(s)) = n(t-s) Var( 1/\sqrt{n} X\_^{(n)}(t-s))

=t - s,

**5.3 Arithmetic Brownian motion and Geometric Brownian Motion**

1. **Limit of scaled symmetric random walk: Brownian motion**

When n-> \inf, based on Central Limit Theorem

W^{(n)}(t) starting from 0 follows a normal distribution N(0, t)

We view W^{(n)}(t) as a **Brownian motion**

1. **Quadratic variation of Brownian motion**

Given a Brownian motion W\_{t\_j}

Define Q = \sum\_{j=0} ^{n-1} (W\_{t\_{j+1}} - W\_{t\_j})^2

E(Q) = \sum\_{j=0} ^{n-1} E ((W\_{t\_{j+1}} - W\_{t\_j})^2)

= \sum\_{j=0} ^{n-1} (t\_{j+1} – t\_j)

= T

Var(Q) = \sum\_{j=0}^{n-1} 2(t\_{j+1} – t\_{j})

<= 2| C|T (where ||C|| = max|t\_{j+1} –t\_{j})

When |C| = 0,

Var(Q) = 0

This is simply written in differential form **dW dW = dt**

1. **From scaled asymmetric random walk to Geometric Brownian motion**

Consider a scaled asymmetric random walk with factor of \sigma and drift \alpha

W^{(n)}(t) = \sigma1 \* (\sqrt{n} \*M\_{nt} )+ \alpha \*t)

Construct new random variable which satisfies

\delta S / S = W^{(n)}(t) , which is equivalent to

S+ = S(1+\alpha /n + \sigma /\sqrt{n})

S- = S(1+\alpha /n - \sigma /\sqrt{n})

Assuming number of heads is H\_nt, number of tails is T\_nt

Then S\_n(t) = S\_n(0) (1+\alpha /n + \sigma /\sqrt{n})^(H\_nt)

(1+\alpha /n - \sigma /\sqrt{n})^(T\_nt)

Then we can prove when n-> \inf

The distribution of S\_n(t) converges to the distribution of

S(t) = S(0) exp((\alpha – ½ \sigma^2 )t +\sigma W(t))

Where W(t) is a normal random variable with mean 0 and variance t.

1. **Ito Integral**
2. **Ito integral**
3. For simple integrand

Given t\_0, t\_1… t\_n and \Delta t is a constant in between any [t\_k, t\_{k+1}]

I(t) = \sum\_{j=0}^{k-1} \Delta(t\_j)[W(t\_{j+1}) – W(t\_{j}) ]

+ \Delta(t\_k)[W(t) – W(t\_k)]

We can also rewrite I(t) = \int\_0 ^t \Delta(u) dW(u)

1. **Properties of Ito integral**
2. Ito Integral is a martingale
3. Isometry

E I^2(t) = E \int\_0 ^t \Delta^2(u) du

1. **For general integrand**

Choose \Delta\_n(t) such that when n-> \inf

Lim\_{n->\inf} E \int\_0 ^T |\Delta\_n(t) -\Delta(t)|^2 dt =0

Define Ito integral

\int\_0^t \Delta(u) dW(u) = lim\_{n->\inf} \int\_0^t \Delta\_n(u) dW(u)

1. **Ito formula and Geometric Brownian motion**
2. **Ito formula**

Suppose dX\_t = u dt + \sigma dB\_t

If g(t, X) is twice continuously differentiable Y\_t = g(t, X\_t)

d Y\_t = \frac{\partial g}{\partial t} dt + \frac{\partial g}{\partial X} dX\_t

+ ½ \frac{\partial^2 g}{\partial x^2} (d X\_t)^2

Where (dX)^2 = d(X\_t) d(X\_t) = u^2 (dt)^2 + 2 u \sigma dt dB\_t

+ \sigma^2 (dB\_t)^2

dt \* dt = dt \* dB\_t = dB\_t \* dB\_t = 0

dB\_t dB\_t = dt

1. **Example: Geometric Brownian motion**

The geometric Brownian motion satisfies

d N\_t /N\_t= r dt + \sigma dB\_t

to solve this we consider

d(ln N\_t)

= \frac{\partial ln N\_t}{\partial N\_t} d N\_t

+1/2 \frac{\partial^2 ln N\_t}{ \partial N\_t ^2} (dN\_t)^2

= \frac{1}{N\_t} d N\_t + ½ (-\frac{1}{N\_t^2}) (d N\_t)^2

(d N\_t)^2 = r^2 N\_t ^2 (dt)^2 + r N\_t dt \sigma d B\_t

+ \sigma^2 N\_t^2 d^2 B\_t

= 0 + 0 + \sigma^2 N\_t^2 dt

So d(ln N\_t) = \frac{1}{N\_t} d N\_t - ½ \sigma^2 dt

= (r – ½ \sigma^2 ) dt + \sigma d B\_t

ln (N\_t/ N\_0) = (r – ½ \sigma^2 ) t + \sigma B\_t

**N\_t = N\_0 = exp( (r – ½ \sigma^2 ) t + \sigma B\_t)**

1. **Black Scholes Merton equation**

We assume the stock prices following a geometric Brownian motion

1. Stock price:

dS(t) = \alpha S(t) dt + \sigma S(t) dW(t)

1. We have a portfolio X(t) which consists of:

\delta t share of stock \Delta(t) S(t)

And (X(t) - \Delta (t) S(t)) money market account with interest rate r

1. Change of the portfolio with respect to time

d X(t) = \Delta(t) d S(t) + r(X(t) - \Delta(t) S(t)) dt

= r X(t) dt + \delta(t) (\alpha – r )S(t)

+ \Delta (t) \sigma S(t) d W(t)

1. Change of the present value of the portfolio with respect to time

d(e^{-r t} S(t)) = (\alpha – r) e^{- r t}S(t) dt

+ \sigma e^{- r t} S(t) dW(t)

1. With a few steps, we get

d(e^{- r t} X(t)) = \Delta(t) d(e^{- r t} S(t))

1. Assume the option value is c(t,x)

d(e^{-rt } c(t, S(t))

= e^{-rt} [- r c(t, S(t)) + c\_t(t, S(t)) + \alpha S(t) c\_S(t, S(t))

+1/2 \sigma^2 S^2(t) c\_SS(t,S(t))] dt

e^{-rt} \sigma S(t) c\_S(t,S(t)) dW(t)

1. Now equate 5) and 6)

dW(t) term:

\Detla(t) = c\_{S}(t, S(t) )

dt term:

rc(t, S) =

c\_t (t, S(t)) + r S c\_{S} (t, S(t)) + 1/2 \sigma^2 S^2(t) c\_{SS}(t, S(t))

which is known **as Black-Scholes-Merton partial differential equation**

1. **Connection of option pricing between binomial model and BSM model**
2. **Binomial Model**

(1+\alpha + \sigma) S\_0 C^{+}

S\_0

(1+\alpha - \sigma) S\_0 C^{-}

At time 0, start with portfolio x share of stock and y amount of saving account with interest rate r, namely S\_0 x + y

At time t=1

(1+ \alpha + \sigma) S\_0 x + (1 + r) y = C+

(1+ \alpha - \sigma) S\_0 x + (1 + r) y = C-

Solve for x and y

Then S\_0 x + y

\frac{1}{1+r\_1}

[\frac{r - \alpha + \sigma}{2 \sigma} C^{+}

+ \frac{r - \alpha + \sigma}{2 \sigma} C^{-}]

When \alpha = r

S\_0 x + y

\frac{1}{1+r\_1} [1/2 C^{+} + 1/2 C^{-}]

1. Black Scholes model

In P measure: S\_t = S\_0 exp[(\alpha -\sigma^2 /2) t + \sigma W\_t]

In Q measure: S\_t = S\_0 exp[(r -\sigma^2 /2) t + \sigma W\_t]

1. The binomial model converges to Black Scholes model when divide time interval from 0 to 1 into n intervals and let n-> \inf
2. **Monte Carlo Simulation**
3. **Toy model: Monte Carlo steps for calculating \pi**
4. Draw random variables x, y belongs U(-1,1)
5. Define indicator function, which indicates whether (x,y) falls into \pi circle

1\_{\pi} = 1 if(x^2 +y^2 <=1)

0 otherwise

1. The ratio of total number of random variable that falls into the circle to the total number pair is ratio of the area of the circle to the area of the square

\frac{ \sum\_i^{N} 1\_{\pi}}{N} = \pi/4

1. \pi is equal to 4/N \sum\_i^{N} 1\_{\pi}
2. **Proof of 3)**

Based on law of large numbers

\frac{ \sum\_i^{N} 1\_{\pi}}{N} = E(1\_{\pi})

= \int 1\_{pi} dP(x,y)

= \int\_{-1}^{1} \int\_{-1}^{1} 1\_{\pi} p(x)p(y) dx dy

= \int\_{-1}^{1} \int\_{-1}^{1} 1\_{\pi} ½ 1/2 dx dy

=1/4 \int\_{-1}^{1} \int\_{-1}^{1} 1\_{\pi} dx dy

= ¼ \* area of the circle

= \pi/4

Var(1\_{\pi}) < \inf since it is a binomial distribution

So Var(\frac{\sum\_i^{N} 1\_{\pi}}{N}) =1/N Var(1\_{\pi}) = 0

1. **Gaussian Random number generator**

**Give U\_1 U\_2 belong [-1,1]**

1. **Principle**
2. Start from an cdf of joint Gaussian random variable x and y

\int \int exp(-1/2 x^2) exp(-1/2 y^2) dx dy

= \int d \theta \int r exp(-r^2/2) dr

= Integral of pdf of \theta \* Integral of pdf of r,

So if we are able to fine r and \theta,

then by taking x = r cos \theta, y = r sin \theta,

we get x and y

1. To fine \theta is easy as we can take \theta = 2 \pi U\_2
2. Take R = \sqrt{-2 ln U\_1}

The cdf of R is

P(R <= r)

= P(\sqrt{-2 U\_1} <= r)

= P(U\_1 >= exp(-1/2 r^2))

= \int\_{exp(1/2r^2)}^1 du\_1

= 1 – exp(-1/2 r^2)

So the pdf = r exp(-1/2 r^2)

1. Therefore

X = \sqrt{-2 ln U\_1} cos(2\pi U\_2)

Y = \sqrt{-2 ln U\_1} sin(2\pi U\_2)

1. **Efficient way of evaluating cos and sin function**

In most computers, evaluating cos and sine is not quite straightforward

1. We generate new random variable S = U\_1^2 + U\_2^2

P(S<s)

= P(U\_1^2 + U\_2^2 <s)

= \int\int\_{u\_1^2 + u\_2^2} 1 dx dy

= \int\_0 ^\sqrt{s} r dr \*2\pi

= \pi r^2

= \pi s

So pdf of s is 1/pi, it is uniform distribution

1. Now if we let S<1, by taking S that satisfies only U\_1^2 + U\_2^2 <1, then throwing away others, then S is uniform in (0,1)

Then a.4) becomes

X = \sqrt{-2 ln S} cos(2\pi S) =\sqrt{-2 ln S} U\_1/\sqrt{S}

Y = \sqrt{-2 ln S} sin(2\pi S) = \sqrt{-2 ln S} U\_2/\sqrt{S}

1. **Final algorithm**
2. Generate two uniform random variables x y belongs to [-1, 1]
3. If x^2 + y^2 >=1, throw away and regenerate
4. Return x \* \sqrt{-2 log (x^2 + y^2) / x^2 + y ^2}

y \* \sqrt{-2 log (x^2 + y^2) / x^2 + y ^2}

1. **Matrix**
   1. **Matrix multiplication**

Consider a matrix A multiplies by B

A B = C

Then each column of C can be viewed as a linear combination of columns of A with B’s column as coefficients.

1st col of C

= b11\* 1st Col of A + b21\* 2nd Col of A + …

+ bn1\* nst Col of A

Each row of C can be viewed as a linear combination of rows of B with A’s row as coefficients.

1st row of C

= a11\* 1st Row of B + a12\* 2nd Row of B + …

+ a1n\* nst Row of B

* 1. **Determinant**

1. **Determinant definition:**
2. Determinant

|A| = \sum\_{All permutations} \sigma(\pi\_j)a\_{1 j\_1} a\_{2 j\_2}….a\_{n j\_n}

1. Minors:

|A\_(i)(j)|: the submatrix by removing ith row and jth colomn

1. Cofactors:

a\_{ij} = (-1)^{i+j} |A\_(i)(j)|

Adjugate matrix: adj(A) = (a\_{ij})^T

1. **Laplace expansion:**

|A| = \sum\_{i}{n} a\_{ij} a\_{(i)(j)}

Property:

A adj(A) = |A|I

The off-diagonal elements are zero because it is the determinant of matrix with the same two rows(or column).

So A^{-1} = \frac{adj(A)}{|A|}

1. **Diagonal expansion:**

A matrix A+D where D is diagonal matrix with all same element d, then

|A+D| = |A| + \sum\_{i\_1,i\_2….i\_n-1} |A\_{i\_1, ,i\_2….i\_n-1}| d

+….+

\sum\_{i\=j} |A\_(i,j)|d^{n-1} +

\sum\_{i} a\_{i, i} d^{n-1}

d^{n}

This is a polynomial of degree n in d, called characteristic polynomial.

1. **Diagonal expansion example:**

A matrix(n x n) has n in its diagonal and all other elements are 1.

A+D D A

n 1 1 …. 1 n - 1 1 1 1 …. 1

1 n 1 …. 1 n – 1 1 1 1 …. 1

1 1 n …. 1 = n - 1 +

… ….

… ….

1 1 1 …. 1 n – 1 1 1 1 … 1

|A+D - \lambda I| = n (n – 1 - \lambda)^(n-1)

+ (n – 1 - \lambda)^n

= (n – 1 - \lambda)\*(2n – 1 -\lambda)

The first term is to choose n - 1 diagonal element in D, one element in A- \lambda I , the second term is choose all diagonal element in D. All other permutations are zero when we select two or more rows in A since it is a matrix with same rows.

1. **Determinant of Matrix Product**

If A and B are square matrix and conformable for multiplication, then

|AB| = |A||B|

**12.3 Rank**

**a. Definition**

The maximum number of linear dependent vectors(columns of rows) in a matrix is the rank of matrix

**b. Basic fact**

The maximum number of linear dependent rows is the row rank, and the maximum number of linear dependent columns is the column rank, they are equal. See wiki for the proof

**c. Basic properties**

1. rank(AB) <= min(rank(A), rank(B))
2. rank(A+B) <= rank(A) + rank(B)
3. rank(AB) >= rank(A) + rank(B) - n
4. rank(A) = r

PAQ = matrix( I\_r 0)

1. 0

**d. Full Rank Factorization**

A n \*m matrix A with rank = r can be factored as product of two full rank matrices.

A = L\_{n\*r} \* R{r\*m} where L is full column rank and R is of full row rank

**e. Existence of Left and right inverse**

If A is n \* m matrix with n<m. Rank(A) = n. Then A has a right inverse. This means a full row rank matrix has a right inverse.

If A is n \* m matrix with m<n. Rank(A) = m. Then A has a left inverse. This means a full column rank matrix has a left inverse.

* 1. **Eigenvalues and Eigenvectors**

1. **Eigenvectors Properties**
2. For any matrix A, all the eigenvectors associated with distinct eigenvalues are linearly independent.
3. For a symmetric matrix A(or Hermitian matrix A), all the eigenvectors associated with distinct eigenvalues are orthogonal.
   1. **Matrix Factorization**
4. **Diagonal Factorization**

If V = (v1, v2, …vn) is the matrix whose columns correspond to eigenvectors of A, and C is a diagonal matrix whose entries are the eigenvalues corresponding to the columns of V, then we have

Av1 = \lambda1 v1

Av2 = \lambda2 v2

Avn = \lambdan vn

A V = (\lambda1 v1, \lambda2 v2,…..\lambdan vn)= VC

So A = VCV-1

1. **Single value decomposition**

Singular value decomposition takes a rectangular matrix (defined as A, where A is a *n*x *p*matrix). The SVD theorem states:

**A*nxp*= U*nxn* S*nxp* VT*pxp***

Where

**U**T**U** = **I**nxn

**V**T**V** = **I**pxp(i.e. U and V are orthogonal)

Calculating the SVD consists of finding the eigenvalues and eigenvectors of *AAT*and *ATA*. The eigenvectors of *ATA* make up the columns of *V*, the eigenvectors of *AAT*make up the columns of *U*. Also, the singular values in **S** are square roots of eigenvalues from *AAT* or *ATA*.  The singular values are the diagonal entries of the *S*matrix and are arranged in descending order. The singular values are always real numbers. If the matrix *A*is a real matrix, then *U*and *V*are also real.

* + 1. LU decomposition

1. **Motivation**

When we solve a linear system equation Ax=b, if we are able to decompose A into L and U, where L is a lower triangular matrix, U is an upper triangular matrix. A = LU, then Ax=b becomes

LUx=b

Let Ux=y, as both L and U are triangular matrix, we can solve for Ly=b first, then Ux=y second.

1. **Formula**

Decomposition form, LU = A

We update starting from the first column to the last column, for each column we update from the top row to the bottom row. For each element a\_ij, we can write it as a product of l’s row and u’s column. We use a\_ij to update l\_ij(i>j) and u\_ij(i<=j)

for i<j, which are the upper triangular part, namely U part(except diagonal),

for i=j, which are the diagonal part, namely U’s diagonal,

for i>j, which are the lower triangle part, namely L part,

1. **In Place Update**

Why we can do in place update?

In the update formula, each a\_ij is used only once, and we get l\_ij or u\_ij, so we can store the l\_ij or u\_ij in the exact same place where a\_ij is stored. This is in-place update.

1. Partial Pivoting

For each l\_ij, we need to divided by u\_jj,

we never like dividing a really small number. So the trick is, for all rows below, we calculate the divider as if they were in the current row, and the excited thing is we do this by the same formula.

We find the largest divider we can use, then we interchange the rows, divided all the rest by the largest divider.

1. Scaled Partial Pivoting

Notice that fact that if we multiply the row of matrix by a very large number, the solution to the linear system equation never change. So when we choose the largest divider, we simply need to multiply it by the factor 1/(largest element in the row) and compare the scaled the divider to find the largest.

d. Java Code

public static int ludcmp (double[][] a,

final int n,

int[] indx,

Parity d

)

{

double[] vv = new double[n];

d.parity=1;

for (int i=0;i<n;++i)

{

double big=0.0;

for (int j=0;j<n;++j)

{

double temp=Math.abs(a[i][j]);

if(temp>big)

big=temp;

}

if (big == 0.0)

return 0;

vv[i]=1.0/big; //save the scale for each row

}

for (int j=0;j<n;++j)

{

int imax = j; // simply assume matrix\_jj element is the pivot

for (int i=0;i<j;++i)

{

double sum=a[i][j];

for (int k=0;k<i;++k)

{

sum -= a[i][k]\*a[k][j]; // calculate u part

}

a[i][j]=sum;

}

double big = 0.0;

for (int i=j;i<n;++i)

{

//cout << " i " << i << endl;

double sum=a[i][j];

for (int k=0;k<j;++k)

// calculate l part, leave the dividing diagonal later

sum -= a[i][k]\*a[k][j];

a[i][j]=sum;

// find which row of this j col should be used as pivot

double dum=vv[i]\*Math.abs(sum);

if(dum >=big)

{

big=dum;

imax=i; // save the imax as the pivot

}

}

if (j != imax) // if interchanging rows, do it.

{

for (int k=0;k<n;++k)

{

double dum =a[imax][k];

a[imax][k]=a[j][k];

a[j][k]=dum;

}

// interchaning once, permutation of det needs a minus sign

parity = -d.parity;

vv[imax]=vv[j];

}

//save the row number so we know which rows have been interchanged

indx[j]=imax;

if (a[j][j] == 0.0)

{

return 0;

}

// divide by the pivot element for the l parts

if (j != n-1)

{

double dum=1.0/(a[j][j]);

for (int i=j+1;i<n;++i)

a[i][j] \*= dum;

}

}

return 1;

}

* 1. **Special Matrix**

1. **Symmetric Matrix**

A = A^T

1. The eigenvectors associate with distinct eigenvalues (v1, v2, …vn )of a symmetric matrix are orthogonal. So let V be the matrix whose columns are the eigenvectors of A, then VV^T =I, V-1 = VT. If with same eigenvalues, then the eigenvector may not be orthogonal, we can do Gram-Schmit transformation to make it orthogonal.
2. The diagonal factorization of an symmetric matrix is

A = VCVT

= AI

= A \sum\_i vi viT

= \sum\_i ci vi viT

1. Maximum value of A’s quadratic form

xTAx

= xT\sum\_i ci vi viT x

= \sum\_i bT VT vi viTVbci

= \sum\_i bi2 ci

<= max{c\_i} bTb

= max{c\_i} xTx

1. **Hermitian Matrix**

A = A\*(where A\* is the complex conjugate of A)

1. The eigenvalues are real.
2. **Idempotent Matrix A2 = A**
3. Its eigenvalues are either 0 or 1.

Because the eigenvalues of A2 are the squares of the eigenvalues of A.

1. Any vector in the columns space of an idempotent matrix A is an eigenvector of A
2. The number of eigenvalues that are 1 is the rank of an idempotent matrix. tr(A) = rank(A)
3. **Symmetric Positive Definite**

A symmetric positive definite matrix satisfies for any non-zero vector x, x^T A x >0

1. Positive definite matrix is non-singular.

Proof: If A is singular, it means there is a non-zero vector x so that Ax=0. Therefore x^T A x = 0, which is a contradiction.

1. All the eigenvalues are positive.
2. Its leading principal minors are all positive.
3. It has a unique Cholesky decomposition.
   1. **Norm, Condition Number**
4. **Vector L\_p Norm**

**One Norm**

**Two Norm**

**Max Norm**

1. **Matrix Induced Norm**

**One Norm**

**Max Norm**

**Two Norm**

1. **Condition Number**
2. **Fast Fourier Transform**

X\_k = \sum\_{n=0}^{N-1} x\_n e^{-\frac{2\pi i}{N} nk}

**Basis orthogonality**

u\_k = [e^{-\frac{2\pi i}{N} nk} | n = 0, 1, 2,…N]^{T}

u\_k1^T u\_k2\* = N \delta{k1, k2}

**reorgnize X\_k with even index terms and odd index terms**

X\_k = \sum\_{m=0} ^{N/2-1} x\_2m e^{-\frac{2\pi i}{N} 2mk}

Even index parts

+ \sum\_{m=0} ^{N/2-1} x\_{2m+1} e^{-\frac{2\pi i}{N} (2m+1)k}

Odd index parts

= \sum\_{m=0} ^{N/2-1} x\_2m e^{-\frac{2\pi i}{N/2} mk}

(We can view this as Fourier Transform of N/2 even indexed points, where k is 0,1…N/2)

+ e^{-\frac{2\pi i}{N} k}

\sum\_{m=0} ^{N/2-1} x\_{2m+1} e^{-\frac{2\pi i}{N/2} mk}

(We can view this as Fourier Transform of N/2 odd indexed points, where k is 0,1…N/2)

= E\_k + e^{-\frac{2\pi i}{N} k} O\_k

X\_{k+N/2} = E\_k - e^{-\frac{2\pi i}{N} k} O\_k

Now we have divided the FFT of N points to two FFT with N/2 points

Keep going till we reach the size to one, then combine together recursively.

#include *<complex>*

#include *<cstdio>*

#define M\_PI 3.14159265358979323846 *// Pi constant with double precision*

**using** **namespace** std;

*// separate even/odd elements to lower/upper halves of array respectively.*

*// Due to Butterfly combinations, this turns out to be the simplest way*

*// to get the job done without clobbering the wrong elements.*

void separate (complex<double>\* a, int n) {

complex<double>\* b = **new** complex<double>[n/2]; *// get temp heap storage*

**for**(int i=0; i<n/2; i++) *// copy all odd elements to heap storage*

b[i] = a[i\*2+1];

**for**(int i=0; i<n/2; i++) *// copy all even elements to lower-half of a[]*

a[i] = a[i\*2];

**for**(int i=0; i<n/2; i++) *// copy all odd (from heap) to upper-half of a[]*

a[i+n/2] = b[i];

**delete**[] b; *// delete heap storage*

}

*// N must be a power-of-2, or bad things will happen.*

*// Currently no check for this condition.*

*//*

*// N input samples in X[] are FFT'd and results left in X[].*

*// Because of Nyquist theorem, N samples means*

*// only first N/2 FFT results in X[] are the answer.*

*// (upper half of X[] is a reflection with no new information).*

void fft2 (complex<double>\* X, int N) {

**if**(N < 2) {

*// bottom of recursion.*

*// Do nothing here, because already X[0] = x[0]*

} **else** {

separate(X,N); *// all evens to lower half, all odds to upper half*

fft2(X, N/2); *// recurse even items*

fft2(X+N/2, N/2); *// recurse odd items*

*// combine results of two half recursions*

**for**(int k=0; k<N/2; k++) {

complex<double> e = X[k ]; *// even*

complex<double> o = X[k+N/2]; *// odd*

*// w is the "twiddle-factor"*

complex<double> w = exp( complex<double>(0,-2.\*M\_PI\*k/N) );

X[k ] = e + w \* o;

X[k+N/2] = e - w \* o;

}

}

}