

지 도 투 영 법

(국지기상 연속감시 시스템)

2000. 9

기 상 청

정 보 화 담 당 관

지 도 투 영 법

(국지기상 연속감시 시스템)

2000. 9

기 상 청

정 보 화 담 당 관

확 인 자	정보화담당관 이 희 훈
작 성 자	기상연구사 이 정 환
작 성 일	2000. 9. 1

머 리 말

세계 각 지점에서 관측한 기상정보를 지도위에 기입하여 일기도를 만들면서부터 기상예보가 시작되었다. 모든 기상정보는 지도 위에서 기입되고 그려져야 그 의미를 해석하기 쉽다. 또한 각종 기상정보도 서로 비교분석하기 위하여 특정한 지도 포맷을 이용하기도 한다. 즉 지도는 기상정보를 종합, 분석하는데 있어서 가장 기본적인 도구인 셈이다.

하지만 기상청에서 사용하는 지도 및 지도투영법에 대한 정의 및 방법이 아직 정리되어 발간된 것이 없어 개인적으로 이해하거나 또는 필요에 따라 그때그때 정의하고 사용하므로 상호연관성이 떨어지고 서로 공유할 기본적인 자료가 부족하여 시간이 지나도 발전하지 못한 면이 있었던 것 같다.

따라서 미흡하지만 기상청에서 사용하고 있는 지도에 대한 논의와 발전을 위한 기본적인 자료로 이용되도록 책을 발간하게 되었다. 이 책에서 사용된 지도 투영법은 지구가 완전한 구를 가정하고 만든 것이다. 다만 지형적 형태를 나타내기 위한 타원체나 지오이드형태의 지구에는 부적합하겠으나, 기상자료 분석에는 주로 대축적 지도가 사용되기 때문에 수백m 정도의 오차 범위에서 사용가능할 것으로 사료된다.

모쪼록 지도투영에 관한 전문적인 지식을 공유하고 보다 발전적인 방향을 모색하기 위하여 이 책이 유용하게 사용되기를 바라면서 기상업무의 정보화에 기초를 다지는 계기가 되기를 희망한다.

2000. 9. 1

정보화담당관 이희훈

목 차

I. 지도 투영법 일반

1. 지구의 형상과 크기	1
2. 지도 투영의 특성	2
3. 공형조건과 공적조건	3
4. Distortion & Magnification	12

II. 투영법들의 특성 및 변환식

1. Mercator projection	13
2. Transveres mercator projection	16
3. Oblique mercator projection	18
4. Miller cylindrical projection	20
5. Equidistant cylindrical projection	22
6. Sinusoidal projection	24
7. Van der grinten projection	26
8. Lambert conformal conic projection	29
9. Lambert azimuthal equal-area projection	31
10. Albers equal-area conic projection	34
11. Stereographic projection	36
12. Orthographic projection	39
13. Azimuthal equidistant projection	42

III. 지도 변환 실무

1. 지도 변환의 실무과정	45
2. 기상청에서 사용중인 지도	50

IV. 지도 변환 프로그램

1. Fortran 버전	60
2. C 버전	79

별첨.

A. 구면삼각형에서의 삼각함수식	101
B. 격자자료 변환	102

I. 지도 투영법 일반

1. 지구의 형상과 크기

가. 지구 구체설

최초 측정 : BC 250년경 Eratosthenes.

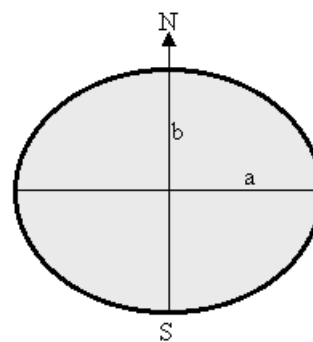
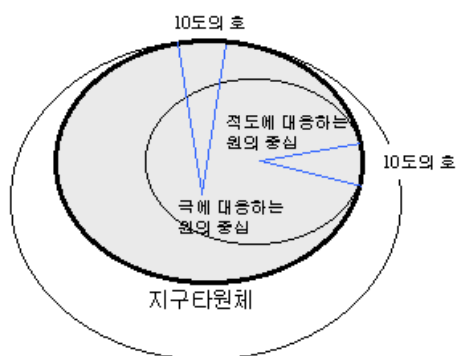
나. 지구타원체

1) 지구구체설의 의문

- J. Richer (1671년, 프랑스 천문학자) : 위도가 높을수록 진자시계가 늦어짐.
- 북극성의 고도가 1도 변하는데 필요한 경도상의 이동거리가 극으로 갈수록 길어짐.

위 도	호의 길이
0° ~ 1°	110.57 km
45° ~ 46°	111.14 km
89° ~ 90°	111.70 km

2) 지구타원체의 편평도



a = 장축 (적도 반경)
b = 단축 (극 반경)

$$\text{편평도} = \frac{a - b}{a}$$

3) 지금까지 사용된 지구타원체의 편평도

지구타원체의 이름	적도반경(km)	편평도	사용국들
Krasovsky(1938)	6,378,245	1/298	소련
International(1924) (Hayford-1909)	6,378,388	1/297	국제적으로 채택
Clarke(1880)	6,378,249	1/293	프랑스, 남아프리카
Clarke(1866)	6,378,206	1/295	북아메리카
Bessel(1841)	6,377,397	1/299	일본, 독일
Airy(1844)	6,377,563	1/299	영국
Everest(1830)	6,377,276	1/301	인도
IUGG(1979)	6,378,137	1/298	북아메리카, 국제적으로 사용하기로 결정됨

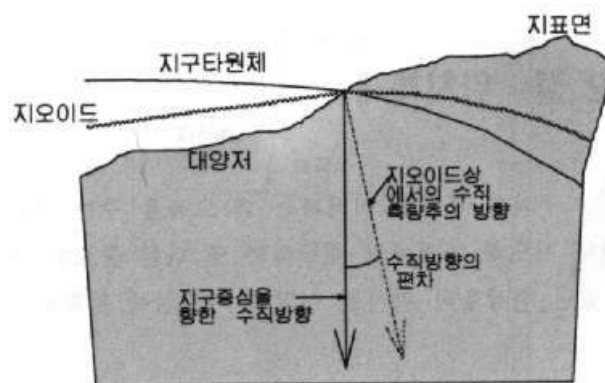
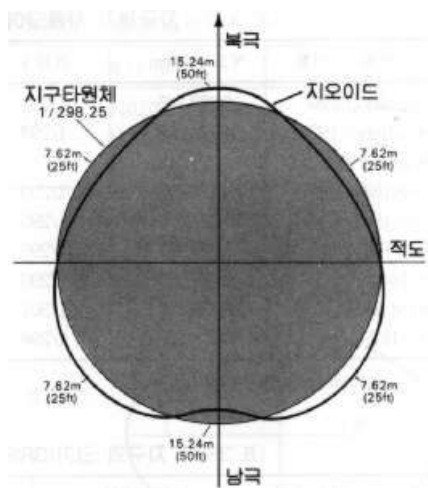
* 국제적으로 GRS80(Geodetic Reference System 1980) 지구타원체를 채택하기로 결정

4) GRS80 지구타원체의 크기

적도 직경	12756.27 km	적도둘레	40085 km
극 직경	12713.50 km	극 둘레	39940 km
편 평 도	1/298	면 적	509,000,000 km ²

다. 지오이드(Goide)

- 중력의 방향에 수직하여 나타나는 이론적인 지표면의 지구타원체
- 지각의 구성요소와 밀도가 장소에 따라 다르기 때문에 지오이드면은 지구타원체보다 낮은 곳(일반적으로 해양)도 있고, 높은 곳(일반적으로 대륙)도 있으며 그 차이는 약 50m 정도이다.



○ 준거타원체(reference ellipsoid)

지오이드는 각 지역마다 조금씩 다르기 때문에 측량오차가 발생할 수 있다. 따라서 지구타원체가 측량의 토대가 되고 지오이드 결정의 기준이 되는 출발 기준(datum)을 정하여 지오이드면과 지구타원체를 일치시켜 놓은 것이 준거타원체이다. 이때 출발기준선은 조차의 영향을 최소화한 평균해수면을 측정하여 채택한다.

지도첩이나 대륙을 한장에 나타내는 소축적의 지도의 경우 구형의 지구모델을 사용해도 별 문제가 없다. 대축적인 지형도를 제작하는 경우는 타원체의 지구모델을 사용하는 것이 보다 정확하다. 아주 세부적인 측지적 조정측량(geodetic control survey)을 필요로 하는 경우에는 지오이드를 사용해야 한다.

일반적으로 대축적, 중축적, 소축적 지도로 구분하나 엄격한 규정이 있는 것은 아니다. 대략 1/50,000 이하인 경우를 대축적지도라 하고 1/1,000,000 이상을 소축적지도라 한다. 일기도의 경우 거의 소축적 지도이다.

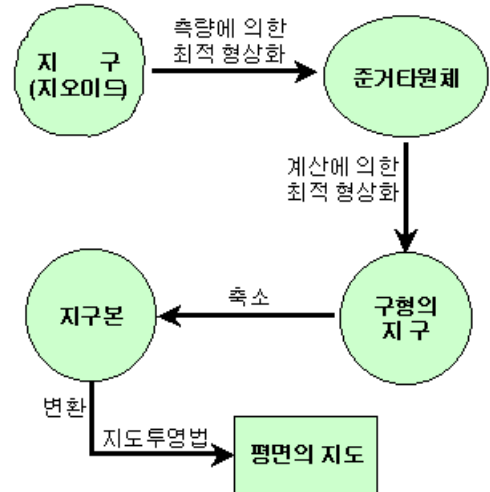
2. 지도 투영

가. 지구본의 특성

지도 투영과정이란 지구 표면을 평면상의 지도로 변환하는 과정을 말한다. 지구의 실제 형상인 지오이드에 가장 부합되는 준거타원체를 설정한 후 일반적으로 이 타원체에 가장 부합되는 면적을 가진 구형의 반경을 계산하여 구형의 지구 모델을 만든다. 지도학자들은 이 구형의 모델을 축소시켜서 지도 투영을 하기 위한 준거지구본(reference globe)을 설정하여 이 지구본을 투영하여 지도를 만드는 것이다.

지구본은 아무런 왜곡을 일으키지 않고 단지 축소시켜서 지구 표면의 실체를 그대로 지도화 한 것이다. 따라서 단지 크기만 변화되었을뿐 상대적인 거리, 방위, 항정선, 대권 등 실제 지구상의 특성을 그대로 유지하고 있다. 이러한 특성을 그대로 가지고 2차원 평면의 지도에 투영하는 것은 불가능하며 필연적으로 특성의 생략 및 왜곡이 있다.

이러한 지도 투영 과정에서 왜곡성을 알기 위해서는 먼저 지구본의 특성을 이해해야 한다. 이러한 지구본의 특성은 아래와 같다.



1) 정형성(Conformality)

- 실제 지표면상에서와 같이 지도상에서도 정각(correct angles)을 유지하는 것을 정형성이라 한다.
- 정각성 또는 정형성을 유지한다는 것은 지도상의 어떤 두 지점간의 나침방향이 지구상에서와 같고, 지도상의 어떤 지역의 형상이 또한 지구상에서의 모양과 닮은 꼴인 것을 의미한다.
- 정형성을 유지하기 위해서는 첫째 경도선과 위도선이 정각으로 교차해야 하고, 둘째 지도상의 어떤 주어진 지점으로부터 모든 방향으로 축척이 동일해야 한다.
- 이 두 조건을 모두 충족하는 것은 지구본상에서만 가능하며, 지도상에서는 포함된 면적이 비교적 적은 지역에서만 가능하다.

2) 정적성(Equivalence)

- 지도상의 어떤 지역의 면적도 지구상에서의 실제 면적과 같은 비례로 나타나는 것을 말한다.
- 정적성을 유지하기 위해서는 한 방향으로의 축척의 변화가 다른 방향으로의 축척의 변화에 의해 상쇄되어야 한다.
- 이 경우 경도선과 위도선이 정각으로 교차하지 못하여 그 모양이 변화된다. 단지 지구본에서만 정형성과 정각성이 동시에 유지된다.

3) 정거성(Equidistance)

- 지도상에서 지점들간의 거리가 지구상에서와 같은 관계를 유지하는 것을 정거성이라 한다.
- 정거성을 유지하기 위해서는 지도상의 두 지점간의 직선간을 연결하는 대권상의 거리가 되어야 한다.
- 특수목적을 위하여 정형성이나 정적성 보다는 정거성을 유지하더라도 모든 지점에서 정거성이 만족되는 것은 아니다
- 대부분 특정 한 지점 또는 두 지점에서 다른 지점들간에만 정거성이 만족된다.

4) 진방위(true azimuth)

- 지도상의 각 지점들간의 방위가 지표면상에서의 방위와 같이 나타날 경우를 진방위라 한다.
- 이를 유지하기 위해서는 지도상에 나타난 두 지점간에 그려진 직선이 두 지점간을 연결하는 대권이여야 하고 또한 진방위이어야 한다. 즉 경도선의 방향이 북극을 나타내는 방향이어야 한다.
- 지도의 중심에서 다른 모든 지점으로의 방향이 진방위를 나타내도록 투영한 방위도법의 경우, 특히 심사도법에서는 두 지점간을 연결하는 대권이 직선으로 나타날 수가 있지만, 이는 한정된 좁은 지역을 나타낼 경우이며 세계 전체를 투영한 지도일 경우 모든 방향이 진방위로 나타낼 수는 없다.

나. 지구본의 속성유지에 따른 투영방법

1) 정형(정각)도법(Conformal projection)

- 지도상의 경도선과 위도선의 교차각도가 지구본상에서와 같이 유지되는 투영법으로 모양이 지구본에서의 모양과 닮은 꼴이다.
- 정각성을 유지하기 위해서는 경도선과 위도선이 정각으로 교차해야하며, 한 지점에서 모든 방향으로의 축척이 같아야 한다. 그러나 이 경우 지도상의 다른 지점간의 축척은 서로 매우 다르다.
- 정각도법에서도 대륙과 같은 넓은 지역에서는 정형성을 유지하기 힘들며, 주로 좁은 지역에서만 정형성이 유지된다.
- 따라서 정형도법은 지표상의 어떤 대상물의 움직임을 위치화하거나 분석하는데 적합하며 해도나 기상도 등에 많이 사용된다.

2) 정적도법(Equal-area projection)

- 지구상에서의 모든 지역간의 면적관계가 지도에서도 그대로 유지되도록 한 투영법
- 이러한 정적성을 유지하기 위해서는 경도선과 위도선을 따라 각기 축척이 조정되어야 한다. 따라서 경도선과 위도선이 정각으로 교차할 수 없으며, 그 모양이 변형된다. 형상, 거리, 방향의 왜곡도는 지도의 주변부로 갈수록 더 심하게 나타난다.
- 정적도법은 지역간의 어떤 현상들의 밀도(인구, 식생 등)를 나타내거나 면적의 관계가 중요한 지도에 많이 사용된다.

3) 정거도법(Equi-distance projectio)

- 지구상에서와 같은 거리관계를 지도상에서도 그대로 유지하도록 투영하는 도법
- 정거성을 유지하기 위해서는 대권상의 거리를 유지해야 하며, 지도상에서 두 지점간의 직선 거리가 지구

상에서의 두 지점간에 대권상의 호가 되어야 한다.

- 그러나 정거성을 유지하는 것이 모든 지점에서 다 해당되는 것이 아니라 한 지점이나 두 지점에서 다른 모든 지점들까지의 거리가 지구상에서와 같다는 것이다. 즉 정거성의 속성은 지도상에 나타난 모든 지점들에 다 충족되는 것은 아니다.
- 정거성을 나타내는 선을 따라 축적이 동일하며, 다른 방향에서는 축적이 다르다.
- 대표적인 도법으로 정거방위도법이 있다.

4) 방위도법(Azimuthal projection)

- 지도상에 나타난 두 지점을 직선으로 연결하였을때 직선의 방향이 진방위가 되도록 투영한 도법이다.
- 방위도법은 정거도법과 마찬가지로 한 중심지점에서부터 다른 모든 지점까지의 방위가 지구상에서와 같도록 유지되는 것이며, 중심지점이 아닌 지점에서부터 다른 지점들간에 방향이 실제 지구상에서와 같다는 것은 아니다.
- 다른 도법과는 달리 방위도법에서는 정적성이나 정형성 또는 정거성을 함께 유지하도록 투영할 수 있다.
- 지표면상에서 두 지점간의 방향관계는 항해하는데 매우 중요하다.

다. 투영법의 투영기준

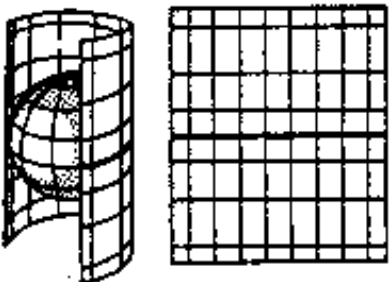
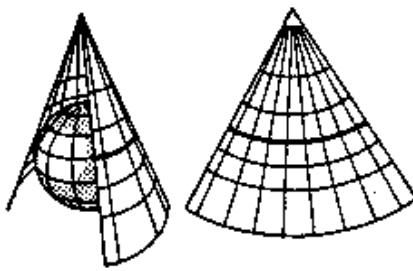
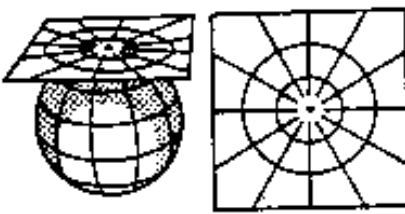
1) 투영면에 따른 분류

지도 투영법은 개념적으로 경도, 위도의 좌표가 그려진 투명한 지구본을 투시하여 비춰진 그림자로부터 지도를 만드는 것으로 볼 수 있다. 이 그림자가 비춰지는 면을 투영면이라 한다.

일반적으로 사용되는 투영면에 따라 아래와 같이 크게 3가지로 분류할 수 있다.

그러나 모든 투영법들이 실제로 광원을 투사하여 그림자를 투영면에 나타내는 방법에 의해서 지도를 구축하는 것은 아니다.

오히려 많은 투영법들이 수학적 방법에 의해서 만들어진다. 그러나 순수히 수학적 방법에 의한 투영법들도 어느 정도는 전통적인 투영면에 의해서 이루어지는 것과 유사하므로 비슷하게 분류할 수 있다.

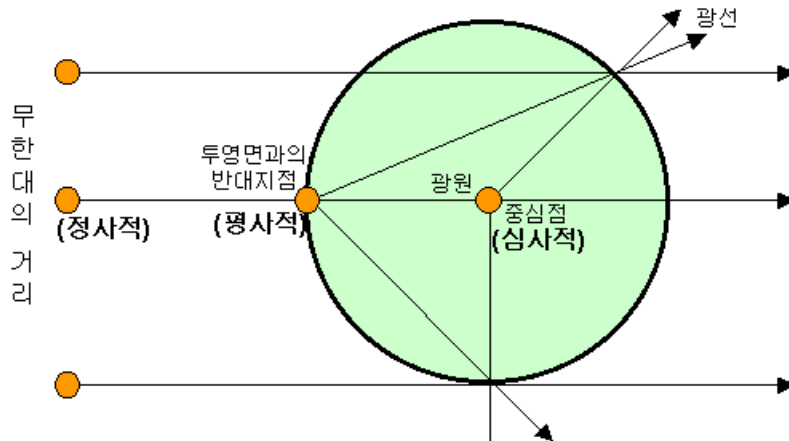
원통도법	원추도법	방위도법
		

2) 광원의 위치에 따른 분류

지도 투영시에 광원이 놓이는 위치에 따라 크게 세가지 유형으로 나눌수 있다.

광원의 위치가 지구본의 중심에 있는 심사적 위치(gnomonic position), 광원의 위치가 투영면이 접하는 지점과 정반대에 평사적 위치(stereographic position), 광원의 위치가 무한대 지점에 있는 정사적 위치(orthographic position)로 분류된다.

광원의 위치가 변화됨에 따라 투영된 격자망의 특성이 달라진다.

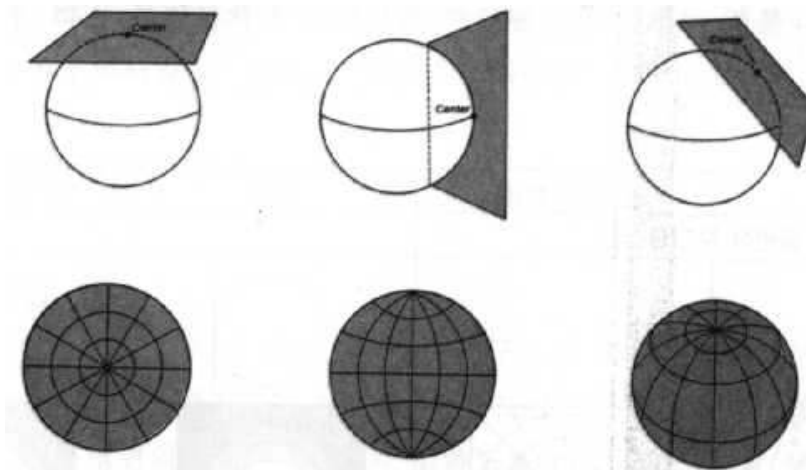


3) 투영면의 접점에 따른 분류

투영면이 적도에 위치한 어떤 지점에 중심을 두고 있는 "적도 중심 투영법", 투영면의 중심이 극과 접한 경우 "극 중심 투영법", 투영면의 중심이 구형의 임의 지점에 접하고 있는 경우 "임의점 중심(oblique aspect) 투영법"이라 한다.

투영면에 따른 분류법에서 보편적인 중심지점을 보면, 방위도법의 경우 극 중심이며, 원통도법은 적도중심이나, 원추도법의 경우는 임의점 중심이다. 그러나 원통도법의 경우 적도가 중심이 아니라 90도 회전한 경도선이 투영면의 중심이 되는 횡축중심(transverse aspect) 투영법도 사용되고 있다.

[투영면과의 접점에 따른 투영법의 분류 : 방위도법의 예]

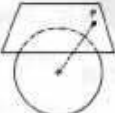
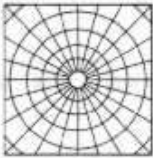



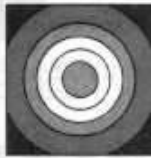

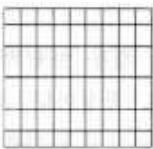





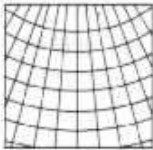






4) 점선/분할선에 따른 분류

투영면의 종류와 관계없이 투영면이 지구본의 어느 한 지점과 접하여 점선을 이루는가 또는 지구본을 교차하여 두 지점과 만나면서 분할선을 이루는가에 따라 투영법을 구분할 수 있다.

점선을 이루도록 투영할 경우 경위도의 좌표체계를 간단히 구축할 수 있는 반면에, 분할선에 의하여 투영할 경우 다소 복잡하지만 지구본과 투영면과의 접하는 부분이 더 많기 때문에 왜곡도를 최소화시킬수 있다는 장점이 있다.

[점선/분할선의 투영방법과 그에 따른 왜곡도의 차이]

종류	경·위선 좌표	점선	분할선
일반적 투영중심점			
 방위도법		 	 
 원통도법		 	 
 원추도법		 	 

출처: Dent, B. D.(1993). n. 44.

마. 일기도용 지도의 조건

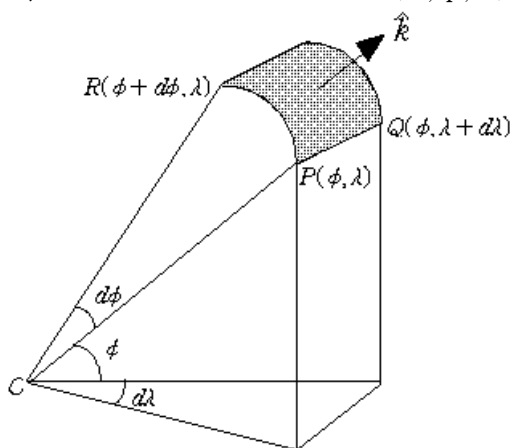
- 연속된 등치선을 그릴 수 있도록 해당 대륙이나 해양이 분열되지 않는 것이 좋음
- 실제 지구상의 모양, 방위(공형성), 면적(공적성)에 대한 오차가 최소한인 지도가 좋음(특히 방위)
- 온도장이 위도에 평행하므로 투영면의 회전축과 지구의 지축을 일치시키는 것이 좋음

3. 공형조건과 공적조건

지도투영이 기하학적인 투영의 개념을 많이 사용하나 실제 변환식은 수학적인 방법을 사용하여 만든다. 이때 많이 사용되는 것이 정형성과 정적성에 해당하는 공형조건과 공적조건이다.

가. 면적소(Area element) 개념

1) 지구의 좌표 - 구면좌표계 (a, ϕ, λ)



지구의 면적소는

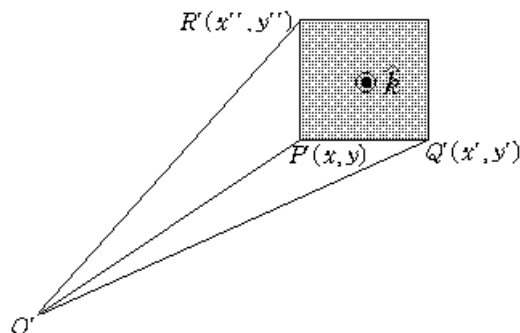
$$\overrightarrow{PQ} = a \cos \phi \, d\lambda \, \hat{\lambda},$$

$$\overrightarrow{PR} = a d\phi \hat{\phi}$$

이므로

$$\begin{aligned}\therefore \Delta \overrightarrow{PQR} &= \overrightarrow{PG} \times \overrightarrow{PR} \\ &= a^2 \cos \phi \, d\lambda \, d\phi \, \hat{k}\end{aligned}$$

2) 지도의 좌표

가) 직각좌표계 (x, y) 

$$P': X = X(\Phi, \Lambda), \quad Y = Y(\Phi, \Lambda)$$

$$Q': X' = X + \frac{\partial X}{\partial \lambda} d\lambda, \quad Y' = Y + \frac{\partial Y}{\partial \lambda} d\lambda$$

$$R': X'' = X + \frac{\partial X}{\partial \Phi} d\Phi, \quad Y'' = Y + \frac{\partial Y}{\partial \Phi} d\Phi$$

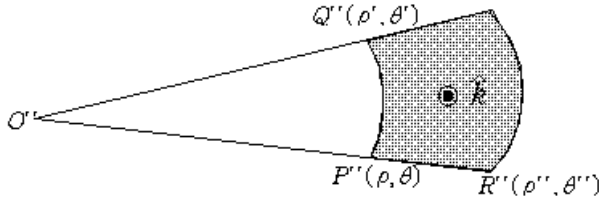
여기서 $\overrightarrow{P'Q'} = \overrightarrow{O'Q'} - \overrightarrow{O'P'} = \frac{\partial x}{\partial \lambda} d\lambda \hat{i} + \frac{\partial y}{\partial \lambda} d\lambda \hat{j}$

$$\overrightarrow{P'R'} = \overrightarrow{O'R'} - \overrightarrow{O'P'} = \frac{\partial X}{\partial \Phi} d\Phi \hat{i} + \frac{\partial Y}{\partial \Phi} d\Phi \hat{j}$$

이므로

$$\therefore \Delta \overrightarrow{P'Q'R'} = \overrightarrow{P'Q'} \times \overrightarrow{P'R'} = \left(\frac{\partial x}{\partial \lambda} \frac{\partial y}{\partial \Phi} - \frac{\partial y}{\partial \lambda} \frac{\partial x}{\partial \Phi} \right) d\lambda d\Phi \hat{k}$$

나) 극좌표계 (r, θ)



$$P'' : r = r(\phi, \lambda), \quad \theta = \theta(\phi, \lambda)$$

$$Q'' : r' = r + \frac{\partial r}{\partial \lambda} d\lambda$$

$$\theta' = \theta + \frac{\partial \theta}{\partial \lambda} d\lambda$$

$$R'' : r'' = r + \frac{\partial r}{\partial \phi} d\phi$$

$$\theta'' = \theta + \frac{\partial \theta}{\partial \phi} d\phi$$

$$\text{여기서 } \overrightarrow{P''Q''} = \frac{\partial r}{\partial \lambda} d\lambda \hat{r} + r \frac{\partial \theta}{\partial \lambda} d\lambda \hat{\theta}$$

$$\overrightarrow{P''R''} = \frac{\partial r}{\partial \phi} d\phi \hat{r} + r \frac{\partial \theta}{\partial \phi} d\phi \hat{\theta}$$

이므로

$$\therefore \Delta \overrightarrow{P''Q''R''} = \overrightarrow{P''Q''} \times \overrightarrow{P''R''} = r \left(\frac{\partial r}{\partial \lambda} \frac{\partial \theta}{\partial \phi} - \frac{\partial \theta}{\partial \lambda} \frac{\partial r}{\partial \phi} \right) d\lambda d\phi \hat{k}$$

나. Equal-Area projection을 위한 등적조건

$$\Delta PQR = \Delta P'Q'R' = \Delta P''Q''R''$$

1) 직각좌표계

$$\left(\frac{\partial x}{\partial \lambda} \frac{\partial y}{\partial \phi} - \frac{\partial y}{\partial \lambda} \frac{\partial x}{\partial \phi} \right) d\lambda d\phi = a^2 \cos \phi d\lambda d\phi$$

$$\therefore \frac{\partial x}{\partial \lambda} \frac{\partial y}{\partial \phi} - \frac{\partial y}{\partial \lambda} \frac{\partial x}{\partial \phi} = a^2 \cos \phi$$

2) 극좌표계

$$\therefore \frac{\partial r}{\partial \lambda} \frac{\partial \theta}{\partial \phi} - \frac{\partial \theta}{\partial \lambda} \frac{\partial r}{\partial \phi} = \frac{a^2}{r} \cos \phi$$

다. Conformal projection을 위한 공형조건

$$\frac{|\overrightarrow{P'Q'}|}{|\overrightarrow{PQ}|} = \frac{|\overrightarrow{P'R'}|}{|\overrightarrow{PR}|} = \frac{|\overrightarrow{Q'R'}|}{|\overrightarrow{QR}|} = K$$

여기서

$$|\overrightarrow{PQ}|^2 = (a \cos \phi d\lambda)^2, \quad |\overrightarrow{PR}|^2 = (a d\phi)^2, \quad |\overrightarrow{QR}|^2 = (a \cos \phi d\lambda)^2 + (a d\phi)^2$$

1) 직각좌표계

$$|\overrightarrow{P'Q'}|^2 = \left(\frac{\partial X}{\partial \lambda} d\lambda\right)^2 + \left(\frac{\partial Y}{\partial \lambda} d\lambda\right)^2$$

$$|\overrightarrow{P'R'}|^2 = \left(\frac{\partial X}{\partial \phi} d\phi\right)^2 + \left(\frac{\partial Y}{\partial \phi} d\phi\right)^2$$

$$|\overrightarrow{Q'R'}|^2 = \left(\frac{\partial X}{\partial \phi} d\phi - \frac{\partial X}{\partial \lambda} d\lambda\right)^2 + \left(\frac{\partial Y}{\partial \phi} d\phi - \frac{\partial Y}{\partial \lambda} d\lambda\right)^2$$

이므로

$$\left(\frac{\partial X}{\partial \lambda} d\lambda\right)^2 + \left(\frac{\partial Y}{\partial \lambda} d\lambda\right)^2 = K^2(a \cos \phi d\lambda)^2 \quad \text{..... ㉠}$$

$$\left(\frac{\partial X}{\partial \phi} d\phi\right)^2 + \left(\frac{\partial Y}{\partial \phi} d\phi\right)^2 = K^2(a d\phi)^2 \quad \text{..... ㉡}$$

$$\begin{aligned} & \left[\left(\frac{\partial X}{\partial \phi}\right)^2 + \left(\frac{\partial Y}{\partial \phi}\right)^2\right] d\phi^2 + \left[\left(\frac{\partial X}{\partial \lambda}\right)^2 + \left(\frac{\partial Y}{\partial \lambda}\right)^2\right] d\lambda^2 \\ & - 2 \frac{\partial X}{\partial \lambda} \frac{\partial X}{\partial \phi} d\phi d\lambda - 2 \frac{\partial Y}{\partial \lambda} \frac{\partial Y}{\partial \phi} d\phi d\lambda = K^2(a \cos \phi d\lambda)^2 + K^2(a d\phi)^2 \end{aligned} \quad \text{..... ㉢}$$

㉠/ $d\phi^2$, ㉡/ $d\lambda^2$ 을 하여 ㉢식을 ㉠식에 대입하면

$$\left(\frac{\partial X}{\partial \lambda}\right)^2 + \left(\frac{\partial Y}{\partial \lambda}\right)^2 = \cos^2 \phi \left\{ \left(\frac{\partial X}{\partial \lambda}\right)^2 + \left(\frac{\partial Y}{\partial \lambda}\right)^2 \right\}$$

㉠, ㉡식을 ㉢식에 대입하면

$$\frac{\partial X}{\partial \lambda} \frac{\partial X}{\partial \phi} + \frac{\partial Y}{\partial \lambda} \frac{\partial Y}{\partial \phi} = 0$$

이때 $\frac{\partial X}{\partial \lambda} \equiv X$, $\frac{\partial Y}{\partial \lambda} \equiv Y$, $\frac{\partial X}{\partial \phi} \equiv A$, $\frac{\partial Y}{\partial \phi} \equiv B$ 라 하면

$$X^2 + Y^2 = \cos^2 \phi (A^2 + B^2)$$

$$XA + YB = 0$$

이다. 이것을 풀면

$$X = \pm B \cos \phi$$

$$Y = \mp A \cos \phi \quad \text{이다. 따라서}$$

$$\therefore \text{공형조건은} \quad \frac{\partial X}{\partial \lambda} = \pm \frac{\partial Y}{\partial \phi} \cos \phi$$

$$\frac{\partial Y}{\partial \lambda} = \mp \frac{\partial X}{\partial \phi} \cos \phi \quad \text{이다.}$$

2) 극좌표계

극좌표인 경우도 앞과 같은 방법으로 하면 다음과 같은 공형조건이 얻어진다.

$$\therefore \text{공형조건은} \quad \frac{\partial r}{\partial \lambda} = \pm \frac{\partial \theta}{\partial \phi} \cos \phi$$

$$r \frac{\partial \theta}{\partial \lambda} = \mp \frac{\partial r}{\partial \phi} \cos \phi \quad \text{이다.}$$

4. Distortion & Magnification

가. Distortion (Image scale, Scale factor) σ

$$\sigma = \frac{\text{투영면상의 거리}}{\text{해당되는 지구상의 실제 거리}}$$

$\sigma > 1$: 투영면이 지구면 밖에 있다

$\sigma = 1$: 투영면이 지구면 밖에 있다

$\sigma < 1$: 투영면이 지구면 안에 있다

σ 를 직각하는 방향의 값(h,k)으로 나타내면

어떤 원통 도법에서 직각좌표계의 경우

$$h = \frac{dy}{Rd\phi}, \quad k = \frac{dx}{R\cos\phi d\lambda}$$

어떤 원뿔 도법에서 극좌표계의 경우

$$h = -\frac{d\rho}{Rd\phi}, \quad k = \frac{n\rho}{R\cos\phi}$$

여기서 n 은 conic constant, ratio of θ to $(\lambda - \lambda_0)$ 이다(원판의 경우 $n=1$).

maximum angular deformation ω 는

$$\sin\omega/2 = \frac{|h-k|}{h+k}$$

이다.

만약 지도가 conformal 이면 $\omega = 0$, $h = k$ 이다. 그리고

만약 지도가 equal-area 이고 위도선과 경도선이 직각으로 만나면 $h = 1/k$ 이다.

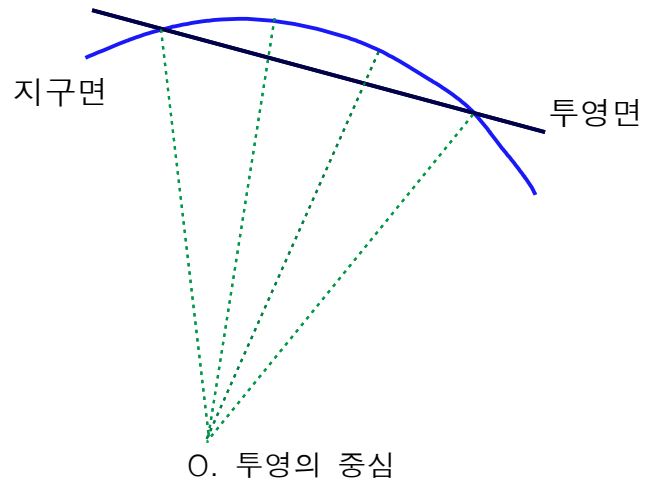
나. Magnification (Map scale, Map factor) m

$$m = \frac{\text{표준위도에서 지도상의 거리}}{\text{해당되는 지구상의 실제 거리}}$$

그러면 지도상의 임의의 점에서의 실제 축척(map scale factor) S 는

$$S = m \sigma$$

이다.



Ⅱ. 투영법 특성 및 변환식

앞의 투영법에 따라 여러 투영법들이 다음과 같이 분류될 수 있다.

○ 원통도법

- Mercator도법 (Mercator Projection)
- 횡축 Mercator도법 (Transverse Mercator Projection)
- Oblique Mercator Projection
- 정적원통도법 (Cylindrical Equal-Area Projection)
- Gall의 원통도법 (Gall Cylindrical Projection)
- Miller의 원통도법 (Miller Cylindrical Projection)
- Equidistant Cylindrical Projection

○ 가상적 원통도법

- Sinusoidal 도법 (Sinusoidal Projection)
- Mollweide 도법 (Mollweide Projection)
- Goode의 Homolosine도법 (Homolosine Projection)
- Eckert 도법 (Eckert Projection)
- Robinson 도법 (Robinson Projection)
- Van Der Grinten Projection

○ 원추도법

- Lambert 정형원추도법 (Lambert Conformal Conic Projection)
- Lambert 정적원추도법 (Lambert Equal-Area Conic Projection)
- Albers 정적원추도법 (Albers Equal-Area Conic Projection)
- 다원추도법 (Polyconic Projection)
- Bonne 도법 (Bonne Projection)

○ 방위도법

- 심사도법 (Gnomonic Projection)
- 평사도법 (Stereographic Projection)
- 정사도법 (Orthographic Projection)
- 정거방위도법 (Azimuthal Equidistant Projection)
- 정적방위도법 (Azimuthal Equal-Area Projection)

이들 중에서 변환식이 비교적 손쉬운 13개 투영법에 대하여 아래에 소개하였다.

1. Mercator projection (메르카토르 도법)¹⁾

특 징

- Cylindrical
- Conformal
- 경도선은 등간격의 직선들이다.
- 위도선은 등간격이 아닌 직선들이다. 적도에서 서로 가장 가깝고, 극에서 경도선을 끊는다.
- Scale은 적도 또는 적도에 대칭인 두 위도선에서 실제와 같다.
- 등사(等斜)곡선은 직선이다.
- 원근(perspective)은 맞지 않다.
- 극점은 무한대에 있다. 극에서 찌그러짐(distortion)이 가장 크다.
- 항해나 운항용으로 사용된다.
- 1569년 Mercator에 의해서 제안되었다.

변 환

❶ 특성

- ▷ 표준위도 = ϕ_s
- ▷ X축 = 적도(동쪽+), Y축 = 중심경도선 λ_0 (북쪽+)
- ▷ 좌표계의 원점의 위경도 = $(0^\circ, \lambda_0)$
- ▷ 적도(표준위도선)에서 축척 = k_0

❷ 위경도 $\Rightarrow (x, y)$

- $x = R k_0 (\lambda - \lambda_0)$
- $y = R k_0 \ln [\tan (\pi/4 + \phi/2)] = R k_0 \tanh^{-1}(\sin \phi)$

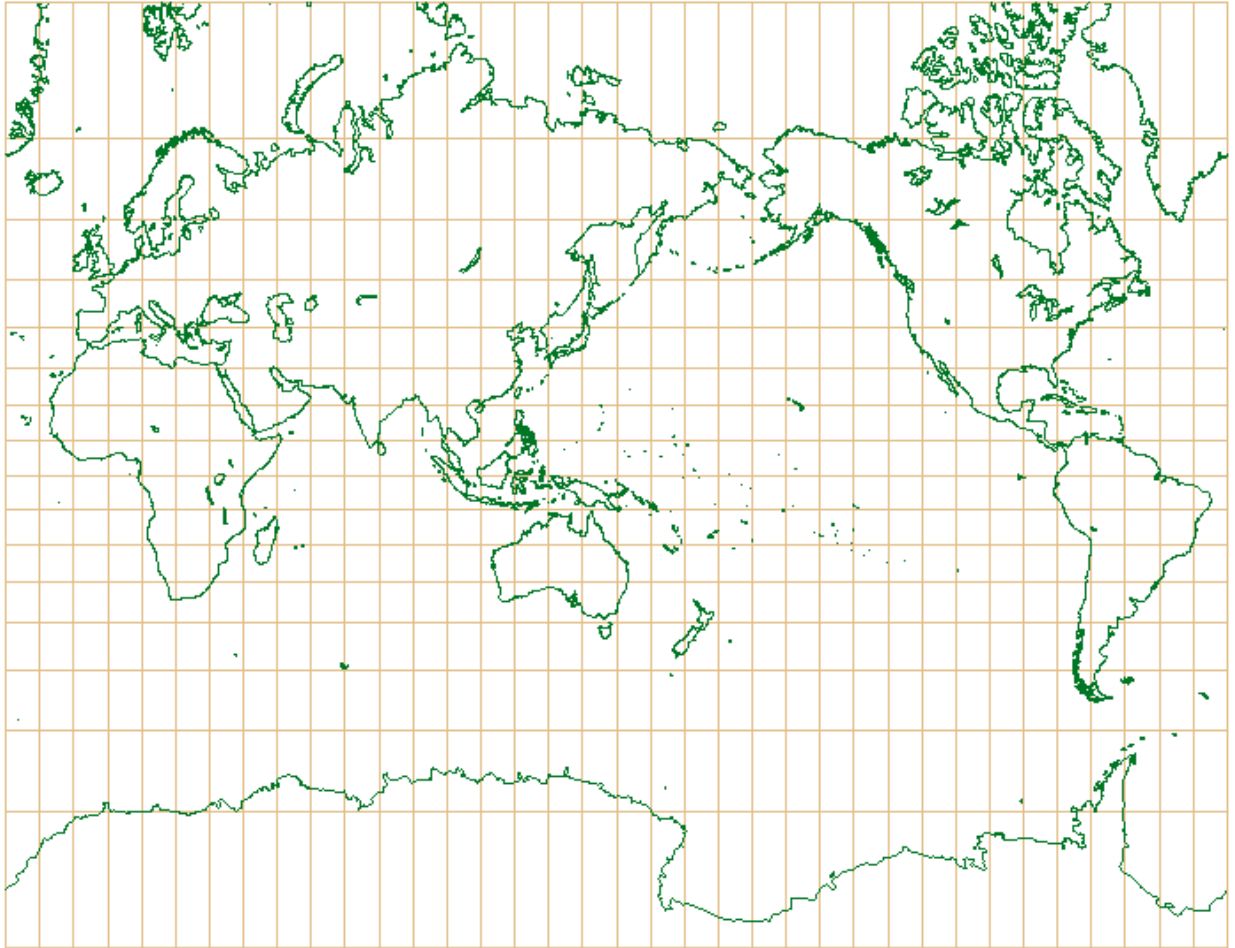
여기서 $h = k = 1/\cos \phi$, $\omega = 0$

표준위도선을 23.5°로 사용하면 $R \rightarrow R \cdot \cos(23.5^\circ)$ 로 바꾸어 사용한다.

❸ $(x, y) \Rightarrow$ 위경도

- $\phi = \pi/2 - 2 \tan^{-1}(e^{-y/Rk_0})$
- $\lambda = x/Rk_0 + \lambda_0$

1) 앞으로의 좌표변환에서 위도와 경도값은 모두 radian이다.



23.5도 표준위도선인 경우 Mercator projection (10도 간격의 위경도선)

2. Transvers mercator projection (횡축 메르카토르도법)

특 징

- Cylindrical (transveres)
- Conformal
- 중심 경도선, 중심 경도선에 90°각도의 경도선들 그리고 적도는 직선이다.
- 다른 경도선과 위도선은 복잡한 곡선이다.
- Scale은 중심경도선 또는 중심경도선에 평행이고 같은 거리에 있는 두 직선 (타원체의 경우 이 직선은 근사적으로 직선이다) 에서 실제와 같다.
- Scale은 중심 경도선에서 90°에서 무한대가 된다.
- 1:24,000 에서 1:250,000 축적의 직사각형의 지도에 널리 사용된다.
- 1772년 Lambert에 의하여 제시되었다.

변 환

❶ 특성

- ▷ Y축 = 중심경도선 λ_0 (북쪽+), X축 = Y축에 수직(동쪽+)
- ▷ 좌표계 원점의 위경도 = (ϕ_0, λ_0)
- ▷ 중심경도선 λ_0 에서의 축적 = k_0

❷ 위경도 $\Rightarrow (x,y)$

$$\bullet x = \frac{1}{2} R k_0 \ln \left[\frac{1+B}{1-B} \right] = R k_0 \tanh^{-1} B$$

$$\bullet y = R k_0 \left[\tan^{-1} \left(\frac{\tan \phi}{\cos(\lambda - \lambda_0)} \right) - \phi_0 \right]$$

$$\text{여기서 } k = k_0 / \sqrt{1 - B^2}, \quad B = \cos \phi \sin(\lambda - \lambda_0)$$

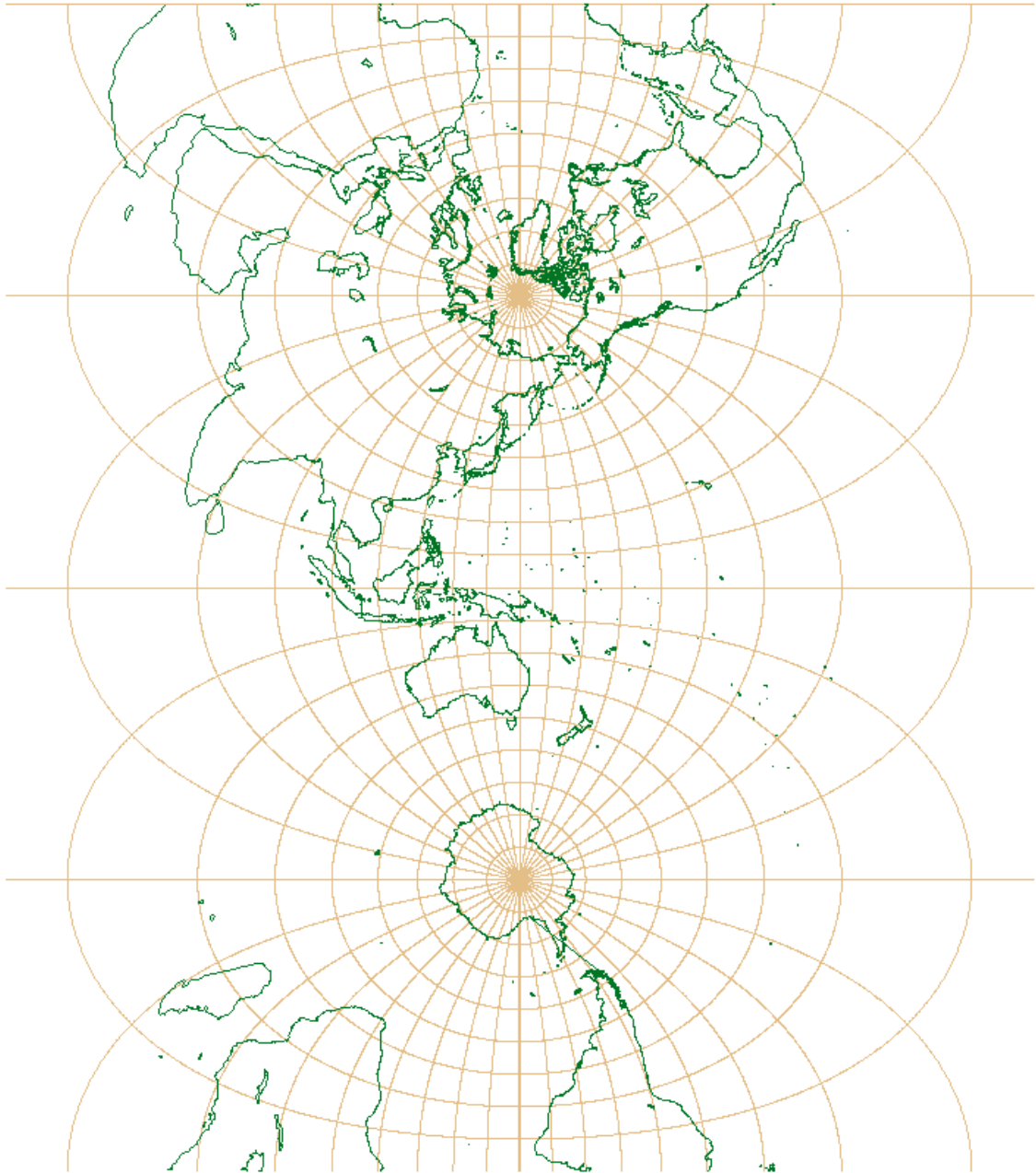
(주의: $B = \pm 1$ 이면 x 는 무한대)

❸ $(x,y) \Rightarrow$ 위경도

$$\bullet \phi = \sin^{-1} \left[\sin D / \cosh \left(\frac{x}{R k_0} \right) \right]$$

$$\bullet \lambda = \lambda_0 + \tan^{-1} \left[\sinh \left(\frac{x}{R k_0} \right) / \cos D \right] \quad (\arctan \text{ 계산은 } \text{atan2} \text{ 함수를 사용})$$

$$\text{여기서 } D = \frac{y}{R k_0} + \phi_0 \quad (\text{using radians})$$



Transveres Mercator Projection

3. Oblique mercator projection

특 징

- Cylindrical (oblique).
- Conformal.
- 180°떨어진 두 경도선만이 직선이다.
- 다른 경도선과 위도선은 복잡한 곡선이다.
- 구면에서의 Scale은 중심선, 즉 빗각(oblique angle)에 가장 큰 대원 또는 중심선에 평행한 두 직선에서 실제와 같다. 타원체인 경우도 비슷하나 이 형태에서 조금 다르다.
- Scale은 중심선에서 90°의 지역에서 무한대이다.
- 위경도선에 비스듬하게 넓은 지역에서 많이 사용된다.
- Rosenmund, Laborde, Hotine, 그리고 여러 다른 사람들에 의하여 1900-50년에 개발되었다.

변 환

❶ 특성

- ▶ Oblique transformation 에서 사용되는 투영 원통면의 축에 해당하는 극의 위경도를 (ϕ_p, λ_p) 라 하면 이 원통면에 대하여 메르카토르도법을 사용하면 된다.
- ▶ 아래에 사용된 좌표계 원점의 위경도 : $(\phi_0, \lambda_0) = (0^\circ, \lambda_p + \pi/2)$
- ▶ 메르카토르도법에서 적도에 해당하는 중심선 위의 두점의 위경도가 $(\phi_1, \lambda_1), (\phi_2, \lambda_2)$ 라 한다면 oblique transformation에서의 극점의 위경도 (ϕ_p, λ_p) 는 다음과 같다.

$$\phi_p = \tan^{-1} \left[-\frac{\cos(\lambda_p - \lambda_1)}{\tan \phi_1} \right]$$

$$\lambda_p = \tan^{-1} \left[\frac{\cos \phi_1 \sin \phi_2 \cos \lambda_1 - \sin \phi_1 \cos \phi_2 \cos \lambda_1}{\sin \phi_1 \cos \phi_2 \sin \lambda_2 - \cos \phi_1 \sin \phi_2 \sin \lambda_1} \right]$$

다른쪽의 극은 $(-\phi_p, \lambda_p \pm \pi)$ 이다.

- ▶ X축 = 중심선(동쪽+), Y축 = X축에 수직

❷ 위경도 $\Rightarrow (x, y)$

$$\bullet x = R k_0 \tan^{-1} \left[\frac{\tan \phi \cos \phi_p - \sin \phi_p \cos(\lambda - \lambda_p)}{\sin(\lambda - \lambda_p)} \right]$$

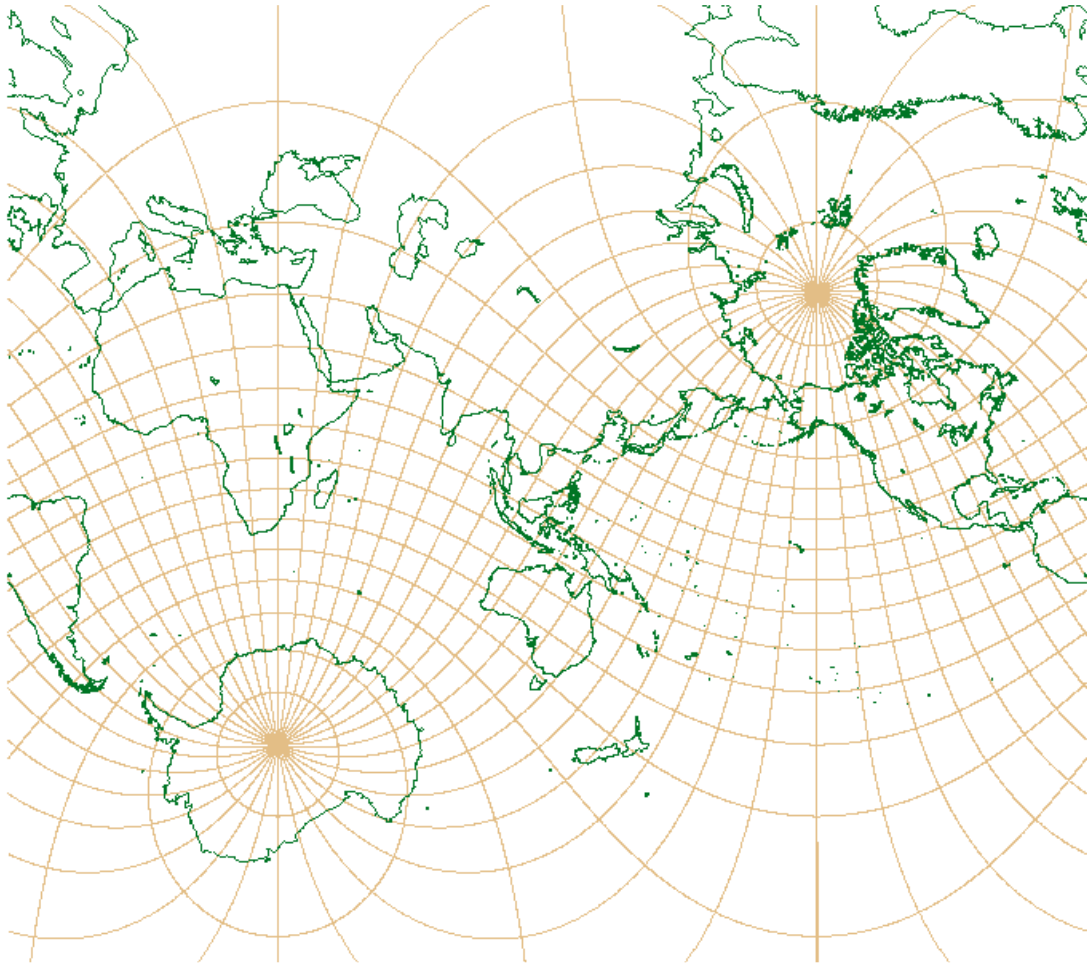
$$\bullet y = \frac{1}{2} R k_0 \ln \left[\frac{1+A}{1-A} \right] = R k_0 \tanh^{-1} A$$

$$\text{여기서 } k = \frac{k_0}{\sqrt{1-A^2}}, \quad A = \sin \phi_p \sin \phi + \cos \phi_p \cos \phi \cos(\lambda - \lambda_p)$$

❸ $(x, y) \Rightarrow$ 위경도

$$\bullet \phi = \sin^{-1} \left[\sin \phi_p \tanh \left(\frac{y}{R k_0} \right) + \cos \phi_p \sin \left(\frac{x}{R k_0} \right) / \cosh \left(\frac{y}{R k_0} \right) \right]$$

$$\bullet \lambda = \tan^{-1} \left[\frac{\sin \phi_p \sin \left(\frac{x}{Rk_0} \right) - \cos \phi_p \sinh \left(\frac{y}{Rk_0} \right)}{\cos \left(\frac{x}{Rk_0} \right)} \right] + \lambda_p + \frac{\pi}{2}$$



Oblique Mercator Projection

4. Miller cylindrical projection

특 징

- Cylindrical.
- Equal-area도, Conformal도 아니다.
- 단지 구면인 경우에만 사용된다.
- 경도선과 위도선들이 모두 직선이며 직각으로 만난다.
- 경도선은 등간격이고 위도선은 적도에서 멀어질수록 더 간격이 넓어진다.
- 극은 하나의 선으로 나타난다.
- 메카토르도법과 다른 원통형 투영법의 혼합이다.
- 세계지도용으로 사용된다.
- 1942년 Miller에 의하여 제안되었다.

변 환

❶ 특성

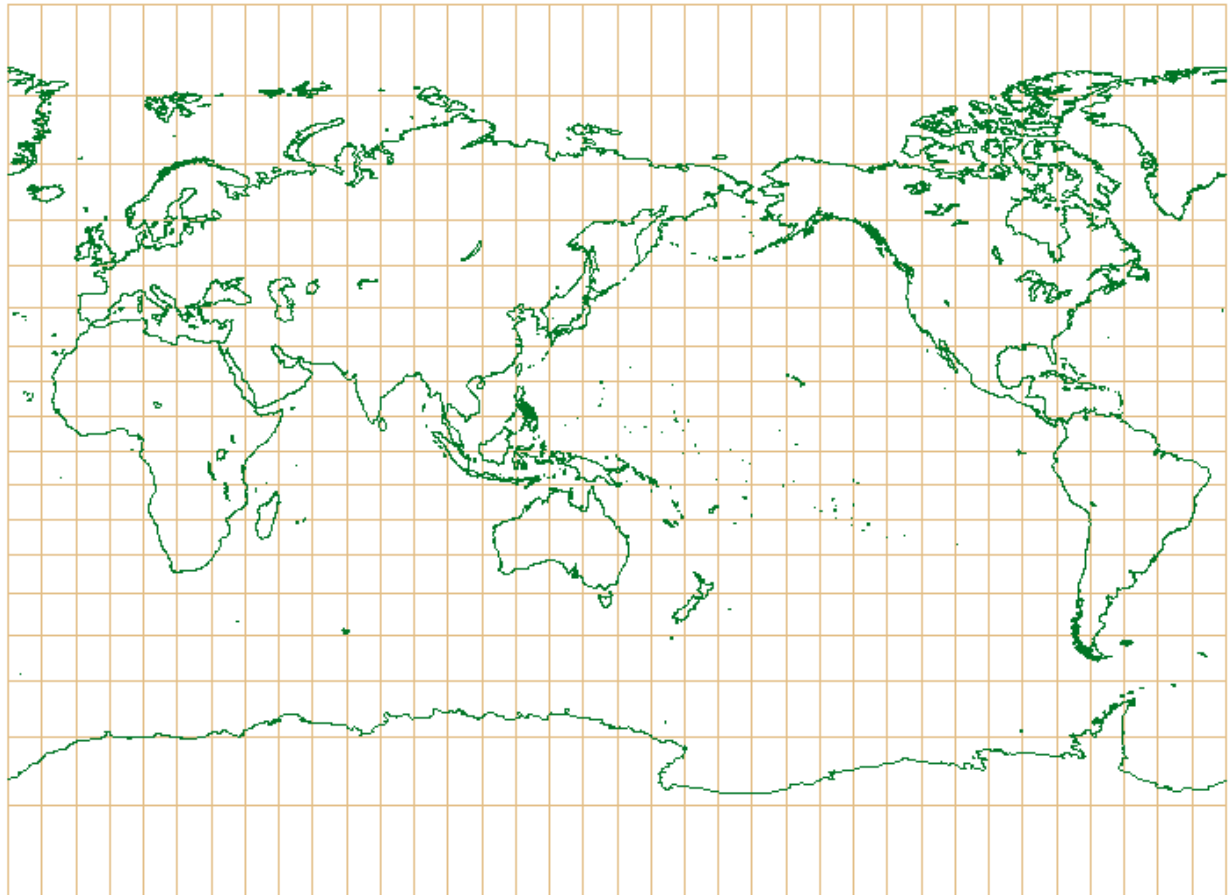
- ▷ X축 = 적도(동쪽+), Y축 = 중심경도선 λ_0 (북쪽+)
- ▷ 아래에 사용된 좌표계 원점의 위경도 : $(\phi_0, \lambda_0) = (0^\circ, \lambda_0)$

❷ 위경도 $\Rightarrow (x, y)$

- $x = R(\lambda - \lambda_0)$
- $y = R [\ln(\tan(\pi/4 + 0.4\phi))] / 0.8 = R [\tanh^{-1}(\sin 0.8\phi)] / 0.8$
 여기서 $h = 1/\cos(0.8\phi)$, $k = 1/\cos\phi$
 $\sin\omega/2 = (\cos 0.8\phi - \cos\phi) / (\cos 0.8\phi + \cos\phi)$

❸ $(x, y) \Rightarrow$ 위경도

- $\phi = 2.5 \tan^{-1}(e^{0.8y/R}) - 5\pi/8 = \sin^{-1}[\tanh(0.8y/R)] / 0.8$
- $\lambda = \lambda_0 + x/R$



Miller Cylindrical Projection

5. Equidistant cylindrical projection

특 징

- Cylindrical
- Equal-area도, Conformal도 아니다.
- 경도선과 위도선이 모두 등간격의 직선이며 서로 직각으로 만난다.
- 극이 직선으로 표시된다.
- 세계지도나 지역지도로 사용된다.
- 매우 간단한 구조이다.
- 단지 구면에서만 사용된다.
- Eratosthenes(B.C.)나 Marinus(A.D.100)에 의해서 제안되었다.

변 환

❶ 특성

- ▷ 표준위도 : ϕ_1
- ▷ X축 = 적도(동쪽+), Y축 = 중심경도선 λ_0 (북쪽+)
- ▷ 아래에 사용된 좌표계 원점의 위경도 : $(\phi_0, \lambda_0) = (0^\circ, \lambda_0)$

❷ 위경도 $\Rightarrow (x, y)$

$$\bullet x = R(\lambda - \lambda_0) \cos \phi_1$$

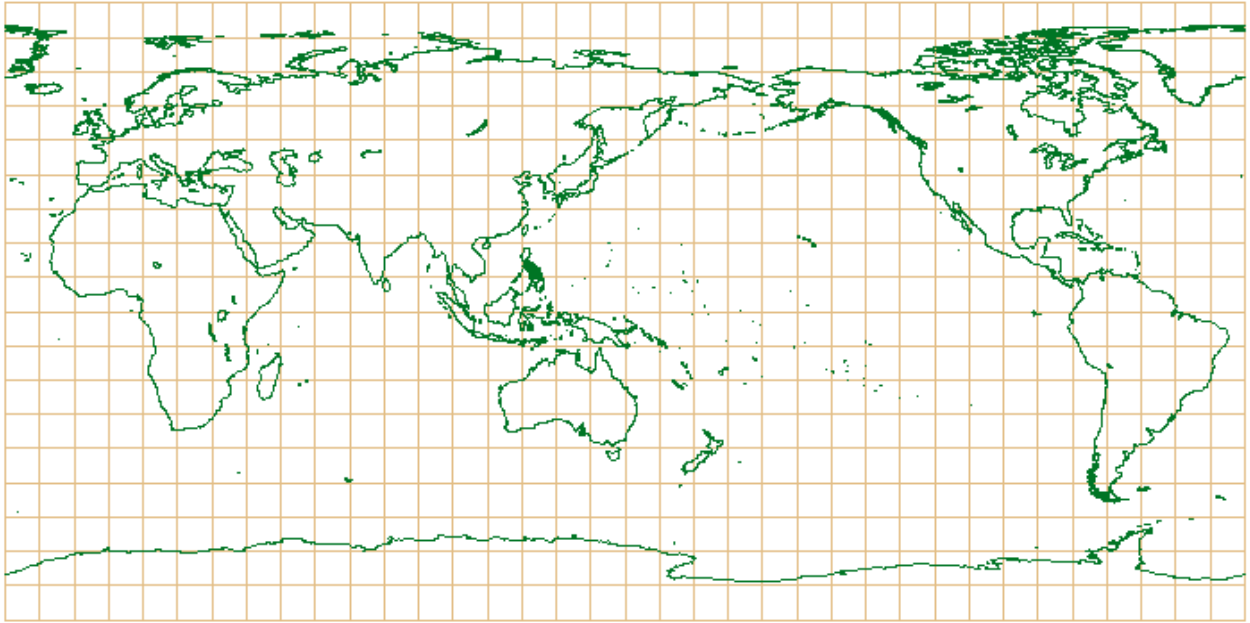
$$\bullet y = R \phi$$

여기서 $h = 1$, $k = \cos \phi_1 / \cos \phi$

❸ $(x, y) \Rightarrow$ 위경도

$$\bullet \phi = y / R$$

$$\bullet \lambda = \lambda_0 + x / (R \cos \phi_1)$$



Equidistant Cylindrical Projection

6. Sinusoidal projection

특 징

- Pseudocylindrical projection
- Equal-area
- 중심경도선은 직선이고, 다른 모든 경도선은 sinusoidal 곡선이다.
- 위도선은 등간격의 직선들이다.
- Scale은 모든 위도선과 중심경도선에서 실제와 같다.
- 남아메리카와 아프리카용 지도로 사용된다.
- 16세기 중반 이후로 사용되어왔다.

변 환

❶ 특성

- ▷ 아래에 사용된 좌표계 원점의 위경도 : $(0^\circ, \lambda_0)$
- ▷ X축 = 적도(동쪽+), Y축 = 중심경도선 λ_0 (북쪽+)

❷ 위경도 $\Rightarrow (x, y)$

- $x = R(\lambda - \lambda_0) \cos \phi$
- $y = R\phi$

여기서 $h = \sqrt{1 + (\lambda - \lambda_0)^2 \sin^2 \phi}$, $k = 1.0$, $\theta' = \sin^{-1}(1/h)$

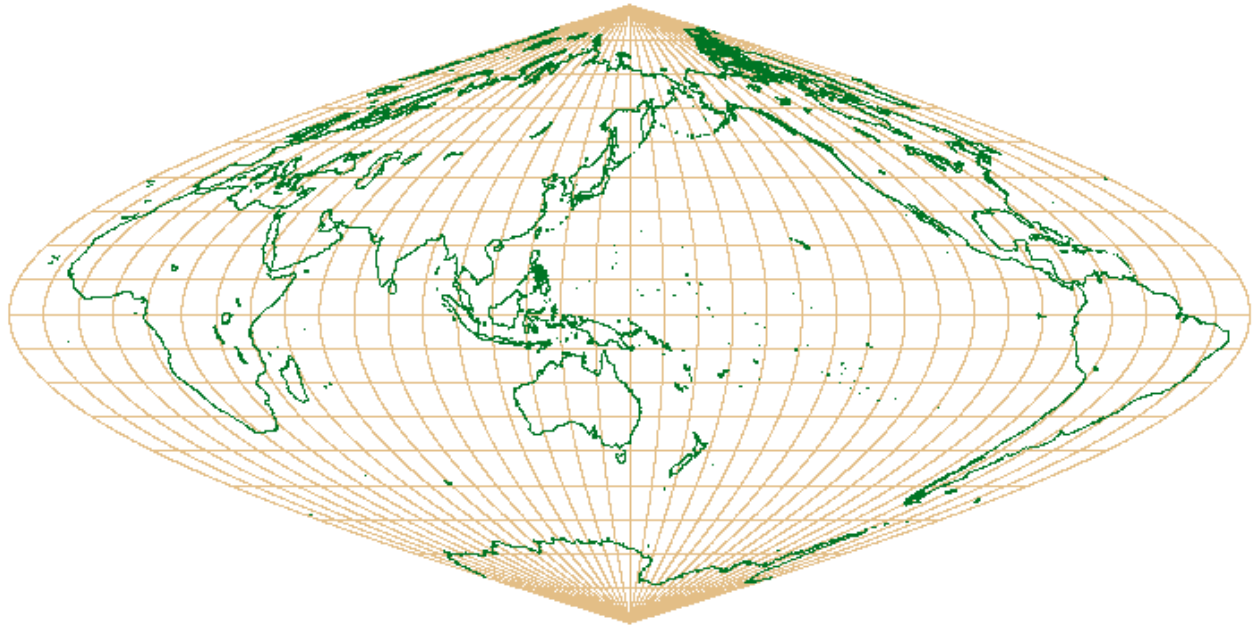
$$\omega = 2 \tan^{-1} [1/2(\lambda - \lambda_0) \sin \phi]$$

이때 θ' 는 지도에서 위도와 경도선이 만나는 각도이다.

❸ $(x, y) \Rightarrow$ 위경도

- $\phi = y/R$
- $\lambda = \lambda_0 + \frac{x}{R \cos \phi}$

여기서 $\phi = \pm 90^\circ$ 인 경우는 계산할 수 없다.



Sinusoidal Projection

7. Van der grinten projection

특 징

- Equal-Area도, Conformal도 아니다.
- 모든 지구면이 하나의 원 안에 모두 나타난다.
- 중심위도선과 적도는 직선이고 다른 위도선과 경도선은 원호이다.
- A curved modification of the Mercator projection, with great dirtortion in the polar areas.
- Scale은 적도에서 실제와 같다.
- 세계지도용으로 사용된다.
- 단지 구면에서만 사용된다.
- 1904년 van der Grinten에 의하여 제안되었다.

변 환

❶ 특성

- ▷ 좌표계 원점의 위경도 : $(0^\circ, \lambda_0)$
- ▷ X축 = 적도(동쪽+), Y축 = 중심경도선 λ_0 (북쪽+)

❷ 위경도 $\Rightarrow (x, y)$

$$\bullet x = \pm \pi R \frac{A(G - P^2) + \sqrt{A^2(G - P^2)^2 - (P^2 + A^2)(G^2 - P^2)}}{P^2 + A^2}$$

$$\bullet y = \pm \pi R \sqrt{1 - \left(\frac{x}{\pi R}\right)^2 - 2A \left|\frac{x}{\pi R}\right|}$$

$$\text{여기서 } A = |\pi/(\lambda - \lambda_0) - (\lambda - \lambda_0)/\pi|/2$$

$$G = \frac{\cos \Theta}{\sin \Theta + \cos \Theta - 1}$$

$$P = G \left(\frac{2}{\cos \Theta} - 1 \right)$$

$$\Theta = \sin^{-1} \left| \frac{2\phi}{\pi} \right|$$

$$\text{if } \phi = 0^\circ: x = R(\lambda - \lambda_0), \quad y = 0$$

$$\text{if } \lambda = \lambda_0: x = 0, \quad y = \pm \pi R \tan(\Theta/2)$$

❸ $(x, y) \Rightarrow$ 위경도

$$\text{if } x=0^\circ: \phi = \frac{\pi \sin \Theta}{2}, \quad \Theta = 2 \tan^{-1} \left(\frac{y}{\pi R} \right)$$

$$\lambda = \lambda_0$$

$$\text{if } y=0^\circ: \phi = 0$$

$$\lambda = \lambda_0 + x/R$$

if $x \neq 0, y \neq 0$:

$$\text{경도는 } \lambda = \lambda_0 \pm \pi [-A + \sqrt{A^2 + 1}]$$

$$A = \frac{1 - \left(\frac{x}{\pi R}\right)^2 - \left(\frac{y}{\pi R}\right)^2}{2 \left| \frac{x}{\pi R} \right|}$$

위도계산은 반복법을 사용하여 계산하여야 한다. 그 초기값을 ϕ_1 이라 하면

$$\phi_1 = 2 \tan^{-1} \left| \frac{y}{\pi R} \right|$$

if $\left| \frac{y}{\pi R} \right| < 0.7$:

$$\Theta = \sin^{-1}(2\phi_1/\pi)$$

$$G = \cos \Theta / (\sin \Theta + \cos \Theta - 1)$$

$$D = \frac{|y/(\pi R)|}{\sin \Theta} - \frac{1}{1 + \cos \Theta}$$

$$H = 2 - \sin \Theta$$

$$J = \tan(\Theta/2)$$

$$\phi_1 = \phi_1 - \frac{[(x/(\pi R))^2 + (y/(\pi R))^2 - 2DGH - J^2]0.25\pi \cos \Theta}{GH \left[\frac{|y/(\pi R)| \cos \Theta}{1 - \cos \Theta} + J \right] + DG \left[\frac{1 + 2\cos^2 \Theta}{\cos \Theta} + H \frac{\cos \Theta - \sin \Theta}{\sin \Theta + \cos \Theta - 1} \right] - \mathcal{K}(J^2 + 1)}$$

if $\left| \frac{y}{\pi R} \right| \geq 0.7$:

$$\phi_1 = \frac{|y| - y_1}{\pi R - y_1} \left(\frac{\pi}{2} - \phi_1 \right) + \phi_1$$



Van der grinten Projection

8. Lambert conformal conic projection

특 징

- Conic
- Conformal
- 위도선은 지도의 중심에서 더 조밀한 비등간격의 동심원이다.
- 경도선은 같은 원의 등간격의 반지름으로 나타나며 위도선과 직각으로 만난다.
- Scale은 두개(또는 한개)의 표준위도선을 따라서는 실제와 같다.
- 표준위도선들이 있는 반구의 극은 한 점으로 표시되며 다른 쪽의 극은 무한대로 간다.
- 동서로 긴 지역의 지도로 사용된다.
- 1772년 Lambert에 의하여 제안되었다.

변 환

❶ 특성

- ▷ 표준위도 : ϕ_1, ϕ_2 (일반적으로 30도, 60를 많이 사용)
- ▷ X축 = 적도(동쪽+), Y축 = 중심경도선 λ_0 (북쪽+)
- ▷ 아래에 사용된 좌표계 원점의 위경도 : $(0^\circ, \lambda_0)$

❷ 위경도 $\Rightarrow (x, y)$

- $x = \rho \sin \Theta$
- $y = \rho_0 - \rho \cos \Theta$

여기서 $\rho = RF / \tan^n(\pi/4 + \phi/2)$, $\Theta = n(\lambda - \lambda_0)$

$$h = k = \frac{\cos \phi_1 \tan^n(\pi/4 + \phi_1/2)}{\cos \phi \tan^n(\pi/4 + \phi/2)}$$

❸ $(x, y) \Rightarrow$ 위경도

- $\phi = 2 \tan^{-1}(RF/\rho)^{1/n} - \pi/2$
- $\lambda = \Theta/n + \lambda_0$

여기서 $\rho = \pm \sqrt{x^2 + (\rho_0 - y)^2}$, n 의 부호에 따라

$$\Theta = \tan^{-1}[x/(\rho_0 - y)]$$

(ϕ 는 atan 함수사용, Θ 는 atan2를 사용한다.

n 이 음수이면 식에 대입하기 전에 x, y, ρ_0 의 부호를 바꾼다.)

위에서 $\rho_0 = RF / \tan^n(\pi/4 + \phi_0/2)$

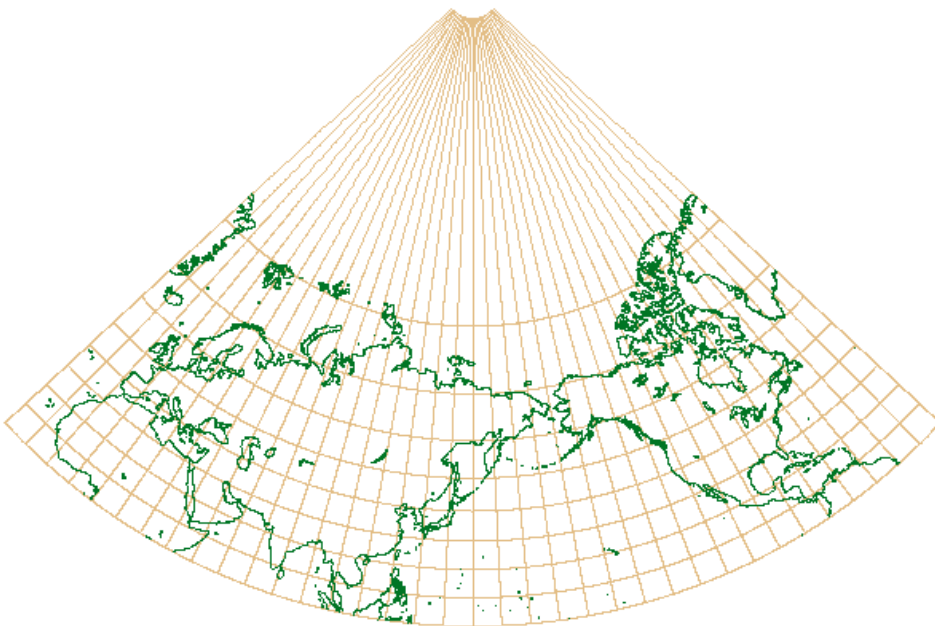
$$F = \cos \phi_1 \tan^n(\pi/4 + \phi_1/2) / n$$

$$n = \frac{\ln(\cos \phi_1 / \cos \phi_2)}{\ln[\tan(\pi/4 + \phi_2/2) / \tan(\pi/4 + \phi_1/2)]}$$

표준위도 = 30N, 60N



표준위도 = 0N, 60N



표준위도 = 0N, 30N

9. Lambert azimuthal equal-area projection

특 징

- Azimuthal
- Equal-Area
- 투영점이 극인 경우 모든 경도선(다른 투영점의 경우는 중심경도선)와 적도에 투영점이 있는 경우 적도가 직선이다.
- 적도에 투영점이 있는 경우 한쪽 반구의 다른 경도선들과 극에 투영점이 있는 경우에 위도선들은 원호이다.
- 다른 모든 경도선들과 위도선들은 복잡한 곡선이다.
- 원근은 맞지 않다.
- Scale은 중심에서의 거리가 멀어짐에 따라 거리의 함수로 감소한다. 중심에서는 찌그러짐이 없다.
- Scale은 중심에서의 거리가 증가할때 반지름에 수직인 방향으로 증가한다.
- 중심에서의 방향은 구인 경우와 극타원체인 경우에는 맞다.
- 구인 경우, 중심의 반대편의 점은 지도 주위의 하나의 원호로서 나타난다.
- 1772년 Lambert에 의하여 제안되었다.

변 환

● 특성

- ▷ X축=Y축에 직각(동쪽+), Y축=중심경도선 λ_0 (북쪽+)
- ▷ 좌표계 원점의 위경도 : (ϕ_1, λ_0)

● 위경도 $\Rightarrow (x, y)$

$$\bullet x = Rk' \cos \phi \sin (\lambda - \lambda_0)$$

$$\bullet y = Rk' [\cos \phi_1 \sin \phi - \sin \phi_1 \cos \phi \cos (\lambda - \lambda_0)]$$

$$\text{여기서 } k' = 1/h' = \sqrt{2 / [1 + \sin \phi_1 \sin \phi + \cos \phi_1 \cos \phi \cos (\lambda - \lambda_0)]}$$

$$\sin \omega / 2 = - \frac{1 - k'^2}{1 + k'^2}$$

if $\phi_1 = 90^\circ$:

$$x = 2R \sin (\pi / 4 - \phi / 2) \sin (\lambda - \lambda_0)$$

$$y = -2R \sin (\pi / 4 - \phi / 2) \cos (\lambda - \lambda_0)$$

$$\text{이때 } k = 1/h = 1/\cos (\pi / 4 - \phi / 2), \quad \rho = 2R \sin (\pi / 4 - \phi / 2), \quad \Theta = \lambda - \lambda_0$$

if $\phi_1 = -90^\circ$:

$$x = 2R \cos(\pi/4 - \phi/2) \sin(\lambda - \lambda_0)$$

$$y = 2R \cos(\pi/4 - \phi/2) \cos(\lambda - \lambda_0)$$

$$\text{이때 } k = 1/h = 1/\sin(\pi/4 - \phi/2),$$

$$\rho = 2R \cos(\pi/4 - \phi/2), \quad \Theta = \pi - \lambda + \lambda_0$$

$$\text{if } \phi_1 = 0^\circ :$$

$$x = Rk' \cos \phi \sin(\lambda - \lambda_0)$$

$$y = Rk' \sin \phi$$

$$\text{이때 } k' = \sqrt{2/[1 + \cos \phi \cos(\lambda - \lambda_0)]}$$

● (x,y) ⇨ 위경도

$$\bullet \phi = \sin^{-1}[\cos c \sin \phi_1 + (y \sin c \cos \phi_1 / \rho)]$$

• 경도

$$\text{if } \phi_1 \neq 90^\circ: \lambda = \lambda_0 + \tan^{-1} \left[\frac{x \sin c}{\rho \cos \phi_1 \cos c - y \sin \phi_1 \sin c} \right]$$

$$\text{if } \phi_1 = 90^\circ: \lambda = \lambda_0 + \tan^{-1}(x/(-y))$$

$$\text{if } \phi_1 = -90^\circ: \lambda = \lambda_0 + \tan^{-1}(x/y)$$

$$\text{여기서 } \rho = \sqrt{x^2 + y^2}, \quad c = 2 \sin^{-1} \left[\frac{\rho}{2R} \right]$$

(arctan는 atan2 함수를 사용)

$$\rho = 2R \sin(c/2)$$

$$\Theta = \pi - Az = 180^\circ - Az \quad (\text{부록 A. 참조})$$

$$h' = \cos(c/2)$$

$$k' = \cos(c/2)$$

여기서 c : angular distance of the given point from the center of the projection

Az : azimuth east of north

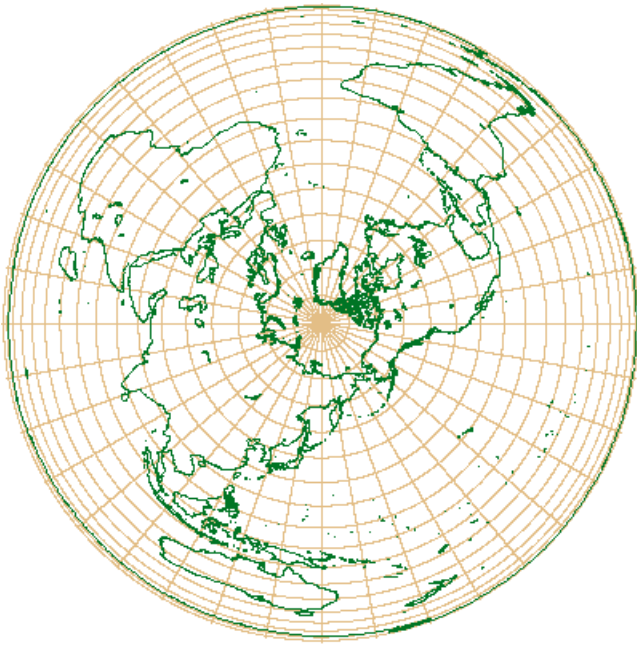
Θ : the polar coordinate east of south

ρ : 투영면의 중심에서의 거리

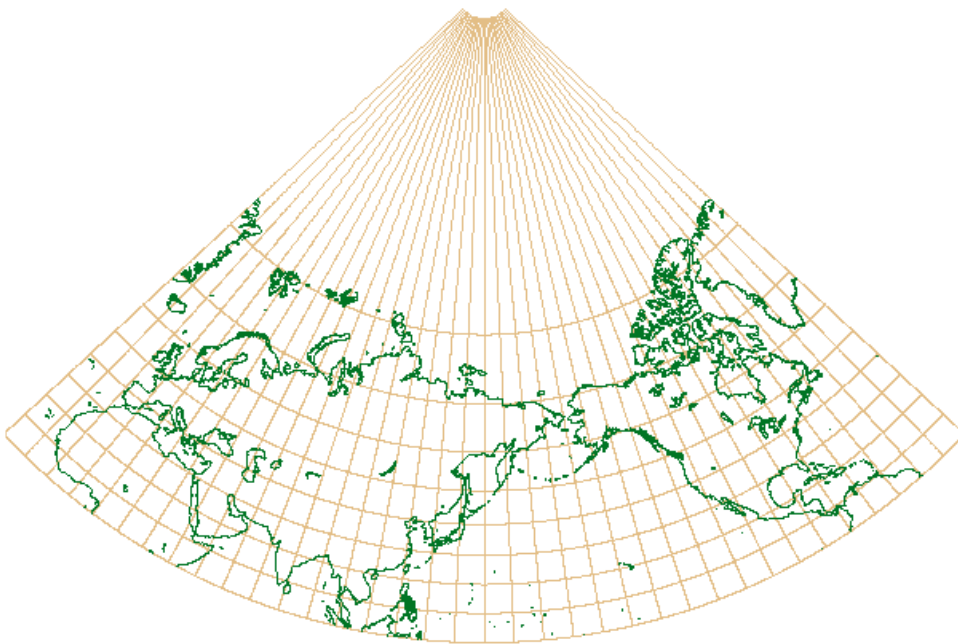
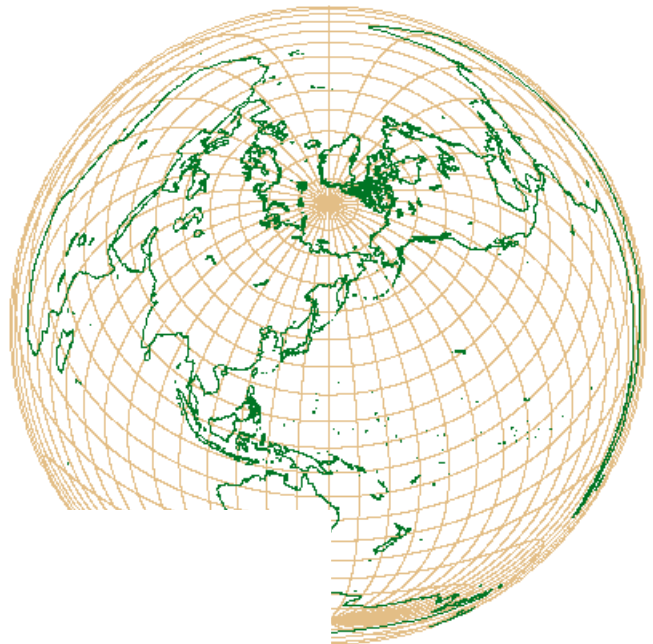
k' : 반지름에 수직인 방향의 scale factor

h' : 반지름 방향의 scale factor

표준위도 = 90N



표준위도 = 45N



표준위도 = 0N

10. Albers equal-area conic projection

특 징

- Conic
- Equal-Area
- 위도선은 지도의 북쪽끝과 남쪽끝 쪽에서 더 조밀한 비등간격의 동심원이다.
- 경도선은 같은 원의 등간격의 반지름으로 나타나며 위도선과 직각으로 만난다.
- 두개(또는 한개)의 표준위도선을 따라서는 축저과 모양에서 찌그러짐이 없다.
- 극은 원으로 나타난다.
- 미국과 같이 동서로 긴 지역에서 equal-area용 지도로 많이 사용된다.
- 1805년 Albers에 의하여 제안되었다.

변 환

● 특성

- ▷ 표준위도 : ϕ_1, ϕ_2
- ▷ Y축 = 중심경도선 λ_0 (북쪽+), X축 = ϕ_0 에서 Y축과 수직(동쪽+)
- ▷ 아래에 사용된 좌표계 원점의 위경도 : (ϕ_0, λ_0)

● 위경도 $\Rightarrow (x, y)$

$$\bullet x = \rho \sin \theta$$

$$\bullet y = \rho_0 - \rho \cos \theta$$

$$\text{여기서 } \rho = R\sqrt{C - 2n\sin\phi} / n, \quad \theta = n(\lambda - \lambda_0)$$

$$h = \cos\phi / \sqrt{C - 2n\sin\phi}$$

● $(x, y) \Rightarrow$ 위경도

$$\bullet \phi = \sin^{-1} \left[\frac{C - (\rho n / R)^2}{2n} \right]$$

$$\bullet \lambda = \lambda_0 + \theta / n$$

$$\text{여기서 } \rho = \sqrt{x^2 + (\rho_0 - y)^2}, \quad \theta = \tan^{-1} [x / (\rho_0 - y)]$$

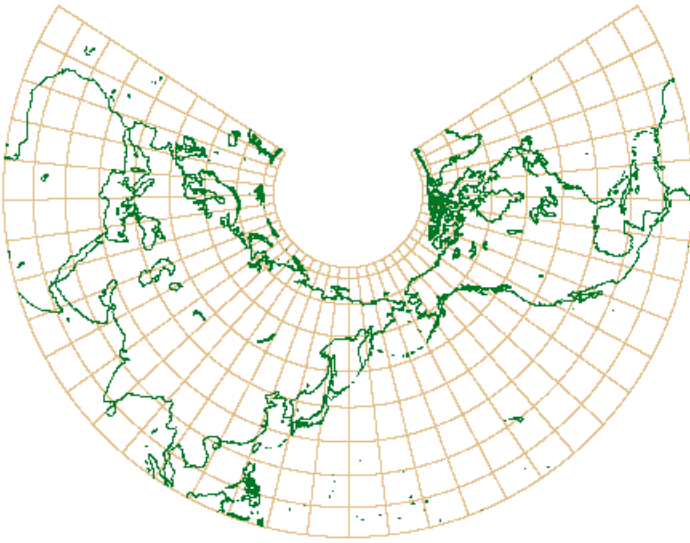
(atan2 함수 사용, n 이 음수이면 x, y, ρ_0 의 부호변환후 계산)

$$\text{위에서 } \rho_0 = R\sqrt{C - 2n\sin\phi_0} / n$$

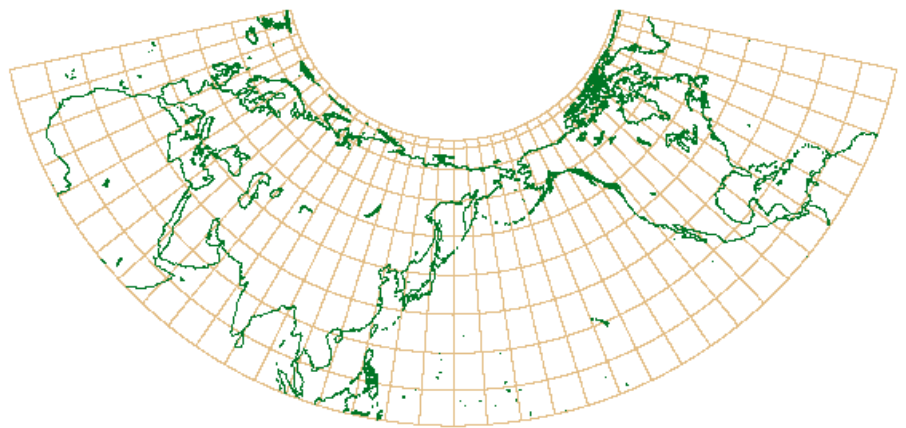
$$C = \cos^2\phi_1 + 2n\sin\phi_1 = 1 + \sin\phi_1\sin\phi_2$$

$$n = (\sin\phi_1 + \sin\phi_2) / 2$$

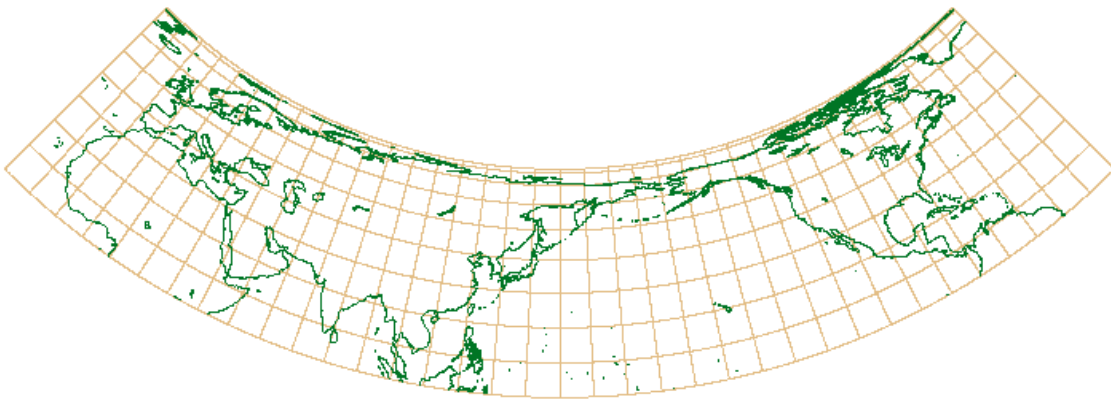
표준위도 = 30N, 60N



표준위도 = 0N, 60N



표준위도 = 0N, 30N



11. Stereographic projection

특 징

- Azimuthal
- Conformal
- 중심경도선과 특별한 위도선은 직선이다. 그리고 모든 경도선은 극에서 만나고, 중심점이 적도에 있는 경우 적도가 직선이다. 다른 모든 경도선과 위도선은 원호이다.
- 구면인 경우에 원근이 맞다.
- 중심점으로부터의 방향은 맞다(ellipsoidal oblique와 equatorial aspects을 제외하고).
- Scale은 투영점에서 멀어질수록 증가한다.
- 투영점의 반대편의 점은 나타나지 않는다.
- 극지방의 지도와 다양한 용도의 지도로 많이 사용된다.
- 기원전 2세기경에 Hipparchus에 의하여 만들어진 것으로 알려져 있다.

변 환

❶ 특성

- ▶ X축=Y축에 직각(동쪽+), Y축=중심경도선 λ_0 (북쪽+)
- ▶ 좌표계 원점의 위경도 : (ϕ_1, λ_0)
- ▶ 투영면이 60N을 지나는 경우, R 대신에 $R(1+\sin 60)/2$ 를 해야한다.

❷ 위경도 $\Rightarrow (x, y)$

- $x = Rk \cos \phi \sin (\lambda - \lambda_0)$
- $y = Rk [\cos \phi_1 \sin \phi - \sin \phi_1 \cos \phi \cos (\lambda - \lambda_0)]$

$$\text{여기서 } k = \frac{2k_0}{1 + \sin \phi_1 \sin \phi + \cos \phi_1 \cos \phi \cos (\lambda - \lambda_0)}$$

if $\phi_1 = 90^\circ$:

$$x = 2Rk_0 \tan (\pi/4 - \phi/2) \sin (\lambda - \lambda_0)$$

$$y = -2Rk_0 \tan (\pi/4 - \phi/2) \cos (\lambda - \lambda_0)$$

$$\text{이때 } k = \frac{2k_0}{1 + \sin \phi}, \quad \rho = 2Rk_0 \tan (\pi/4 - \phi/2), \quad \Theta = \lambda - \lambda_0$$

if $\phi_1 = -90^\circ$:

$$x = 2Rk_0 \tan (\pi/4 + \phi/2) \sin (\lambda - \lambda_0)$$

$$y = 2Rk_0 \tan (\pi/4 + \phi/2) \cos (\lambda - \lambda_0)$$

$$\text{이때 } k = \frac{2k_0}{1 - \sin \phi}, \quad \rho = 2Rk_0 \tan(\pi/4 + \phi/2), \quad \Theta = \pi - \lambda + \lambda_0$$

if $\phi_1 = 0^\circ$:

$$x = Rk \cos \phi \sin(\lambda - \lambda_0)$$

$$y = Rk \sin \phi$$

$$\text{이때 } k = \frac{2k_0}{1 + \cos \phi \cos(\lambda - \lambda_0)}$$

● (x,y) ⇨ 위경도

$$\bullet \phi = \sin^{-1} \left[\cos c \sin \phi_1 + \left(\frac{y \sin c \cos \phi_1}{\rho} \right) \right]$$

• 경도

$$\text{if } \phi_1 \neq 90^\circ : \quad \lambda = \lambda_0 + \tan^{-1} \left[\frac{x \sin c}{\rho \cos \phi_1 \cos c - y \sin \phi_1 \sin c} \right]$$

$$\text{if } \phi_1 = 90^\circ : \quad \lambda = \lambda_0 + \tan^{-1}(x/(-y))$$

$$\text{if } \phi_1 = -90^\circ : \quad \lambda = \lambda_0 + \tan^{-1}(x/y)$$

$$\text{여기서 } \rho = \sqrt{x^2 + y^2}, \quad c = 2 \tan^{-1} \left[\frac{\rho}{2Rk_0} \right]$$

(arctan는 atan2 함수를 사용)

$$\rho = 2R \tan(c/2)$$

$$\Theta = \pi - Az = 180^\circ - Az \quad (\text{부록 A. 참조})$$

$$k = 1/\cos(c/2)$$

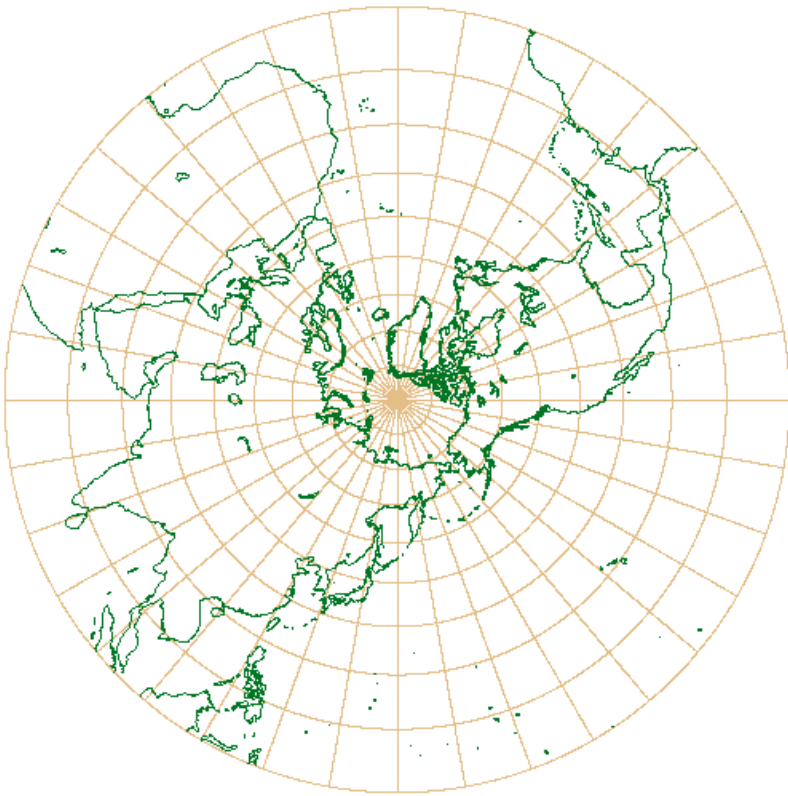
여기서 c : angular distance of the given point from the center of the projection

Az : azimuth east of north

Θ : the polar coordinate east of south

ρ : 투영면의 중심에서의 거리

표준위도 = 90N



표준위도 = 0N



12. Orthographic projection

특 징

- Azimuthal
- 모든 경도선과 위도선은 타원이거나 직선이다.
- 무한히 먼 점에서 원근적으로 바라본 것이므로 실제 지구와 외형이 닮지 않았다.
- 한번에 단지 한쪽의 반구만이 나타난다.
- 나타나는 반구의 가장자리에서 찌그러짐이 가장 크고, 단지 중심점에서만 찌그러짐이 없다.
- 중심점에서의 방향은 맞다.
- Radial scale factor는 중심점에서 멀어질수록 감소한다.
- 위도를 따르는 방향의 Scale은 중심점이 극일때만 실제와 같다.
- 주로 시각적인 모양에서 사용된다.
- 단지 구면에서만 사용된다.
- 2000년전에 그리스인과 이집트인들에 의해서 알려졌다.

변 환

❶ 특성

- ▷ Y축 = 중심경도선 λ_0 (북쪽+)
- ▷ 좌표계 원점의 위경도 : (ϕ_1, λ_0)

❷ 위경도 $\Rightarrow (x, y)$

$$x = R \cos \phi \sin (\lambda - \lambda_0)$$

$$y = R [\cos \phi_1 \sin \phi - \sin \phi_1 \cos \phi \cos (\lambda - \lambda_0)]$$

$$\text{여기서 } h' = \cos c = \sin \phi_1 \sin \phi + \cos \phi_1 \cos \phi \cos (\lambda - \lambda_0)$$

$$k' = 1.0$$

이때 $h' =$ 중심점에서 반지름 방향의 scale factor

$k' =$ 반지름에 직각인 방향의 scale factor

if $\phi_1 = 90^\circ$:

$$x = R \cos \phi \sin (\lambda - \lambda_0)$$

$$y = -R \cos \phi \cos (\lambda - \lambda_0)$$

$$\text{이때 } h = \sin \phi, \quad \rho = R \cos \phi, \quad \theta = \lambda - \lambda_0$$

if $\phi_1 = -90^\circ$:

$$x = R \cos \phi \sin (\lambda - \lambda_0)$$

$$y = R \cos \phi \cos (\lambda - \lambda_0)$$

$$\text{이때 } h = -\sin \phi, \quad \rho = R \cos \phi, \quad \Theta = \pi - \lambda + \lambda_0$$

$$\text{if } \phi_1 = 0^\circ :$$

$$x = R \cos \phi \sin (\lambda - \lambda_0)$$

$$y = R \sin \phi$$

● (x,y) ⇨ 위경도

$$\bullet \phi = \sin^{-1} \left[\cos c \sin \phi_1 + \left(\frac{y \sin c \cos \phi_1}{\rho} \right) \right]$$

• 경도

$$\text{if } \phi_1 \neq \pm 90^\circ: \quad \lambda = \lambda_0 + \tan^{-1} \left[\frac{x \sin c}{\rho \cos \phi_1 \cos c - y \sin \phi_1 \sin c} \right]$$

$$\text{if } \phi_1 = 90^\circ : \quad \lambda = \lambda_0 + \tan^{-1}(x/(-y))$$

$$\text{if } \phi_1 = -90^\circ: \quad \lambda = \lambda_0 + \tan^{-1}(x/y)$$

$$\text{여기서 } \rho = \sqrt{x^2 + y^2}, \quad c = \sin^{-1}(\rho/R)$$

(위에서 atan2 함수를 사용)

$$\rho = R \sin c$$

$$\Theta = \pi - Az = 180^\circ - Az \quad (\text{부록 A. 참조})$$

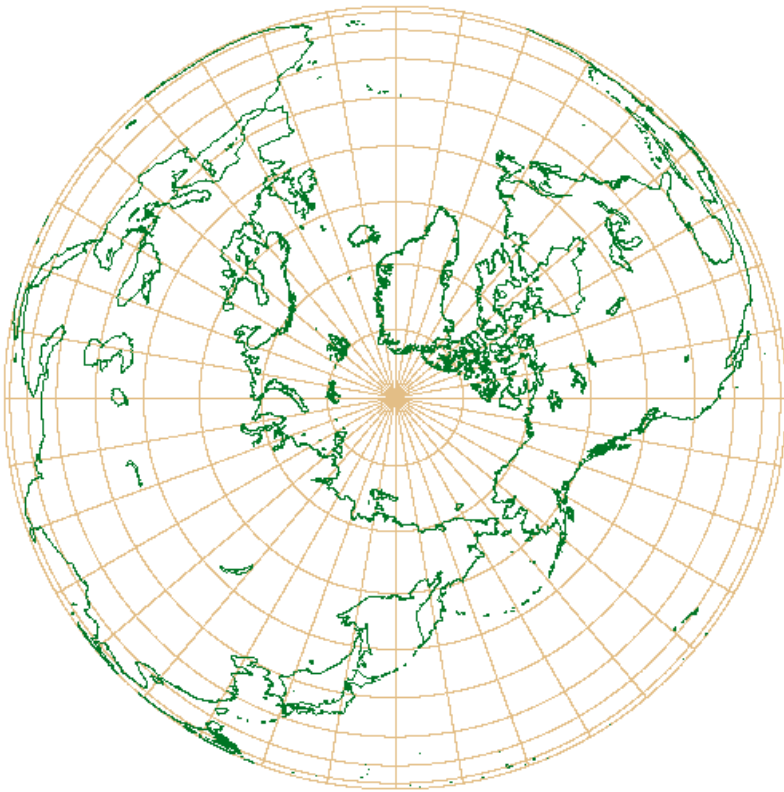
여기서 c : angular distance of the given point from the center of the projection

Az : azimuth east of north

Θ : the polar coordinate east of south

ρ : 투영면의 중심에서의 거리

표준위도 = 90N



표준위도 = 0N



13. Azimuthal equidistant projection

특 징

- Azimuthal
- Conformal도, Equal-area도 아니다.
- 중심에서의 거리는 실제와 같다. 그러나 중심에서 반경을 따른 거리가 아니면 같지 않다.
- 단지 중심점만 찌그러짐이 없다.
- 중심에서의 방향은 실제와 같다(except on some oblique and equatorial ellipsoidal forms).
- 극이 중심점인 경우에는 모든 경도선, 다른 경우에는 중심경도선 그리고 적도에 중심선이 있는 경우에는 적도가 직선이다.
- 극이 중심점인 경우는 위도선이 원호이고 실제 위도간 거리와 같은 간격이다 (따라서 구인 경우는 등간격이 된다).
- 적도에 중심점이 있는 경우에 다른 경도선들은 구인 경우 원이다.
- 다른 경도선과 위도선은 복잡한 곡선이다.
- 원근이 맞지 않다.
- 중심점의 반대편에 있는 점은 구인 경우 지도 주위의 원으로 나타난다.
- 중심점이 극에 있는 경우에는 세계지도나 반구전체를 나타낼때 사용된다.
- 중심점이 다른 위도에 있는 경우는 대륙들의 지도책이나 항공용, 라디오용의 지도에서 사용된다.
- 중심점이 극인 경우는 수세기동안 알려져왔다.

변 환

❶ 특성

- ▷ X축=Y축에 직각(동쪽+), Y축=중심경도선 λ_0 (북쪽+)
- ▷ 아래에 사용된 좌표계 원점의 위경도 : (ϕ_1, λ_0)

❷ 위경도 $\Rightarrow (x,y)$

- $x = Rk' \cos \phi \sin (\lambda - \lambda_0)$
 - $y = Rk' [\cos \phi_1 \sin \phi - \sin \phi_1 \cos \phi \cos (\lambda - \lambda_0)]$
- 여기서 $k' = c / \sin c$, $h' = 1.0$
- $$\cos c = \sin \phi_1 \sin \phi + \cos \phi_1 \cos \phi \cos (\lambda - \lambda_0)$$

if $\phi_1 = 90^\circ$:

$$x = R(\pi/2 - \phi) \sin (\lambda - \lambda_0)$$

$$y = -R(\pi/2 - \phi) \cos (\lambda - \lambda_0)$$

이때 $k = (\pi/2 - \phi) / \cos \phi$, $h = 1.0$, $\rho = R(\pi/2 - \phi)$, $\Theta = \lambda - \lambda_0$

$$\sin \omega/2 = \frac{k' - 1}{k' + 1} = \frac{c - \sin c}{c + \sin c}$$

if $\phi_1 = -90^\circ$:

$$x = R(\pi/2 + \phi) \sin(\lambda - \lambda_0)$$

$$y = R(\pi/2 + \phi) \cos(\lambda - \lambda_0)$$

$$\text{이때 } k = (\pi/2 + \phi) / \cos \phi, \quad h = 1.0, \quad \rho = R(\pi/2 + \phi), \quad \Theta = \pi - \lambda + \lambda_0$$

if $\phi_1 = 0^\circ$:

$$x = Rk' \cos \phi \sin(\lambda - \lambda_0)$$

$$y = Rk' \sin \phi$$

$$\text{이때 } \cos c = \cos \phi \cos(\lambda - \lambda_0)$$

● (x,y) ⇔ 위경도

$$\bullet \phi = \sin^{-1} \left[\cos c \sin \phi_1 + \frac{y \sin c \cos \phi_1}{\rho} \right]$$

• 경도

$$\text{if } \phi_1 \neq 90^\circ: \quad \lambda = \lambda_0 + \tan^{-1} \left[\frac{x \sin c}{\rho \cos \phi_1 \cos c - y \sin \phi_1 \sin c} \right]$$

$$\text{if } \phi_1 = 90^\circ: \quad \lambda = \lambda_0 + \tan^{-1}(x/(-y))$$

$$\text{if } \phi_1 = -90^\circ: \quad \lambda = \lambda_0 + \tan^{-1}(x/y)$$

$$\text{여기서 } \rho = \sqrt{x^2 + y^2}, \quad c = \rho/R$$

(arctan는 atan2 함수를 사용)

$$\rho = R c$$

$$\Theta = \pi - Az = 180^\circ - Az \quad (\text{부록 A. 참조})$$

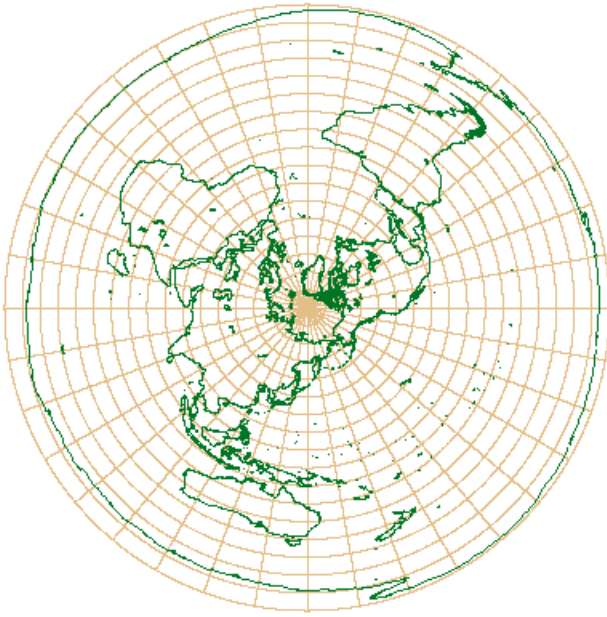
여기서 c : angular distance of the given point from the center of the projection

Az : azimuth east of north

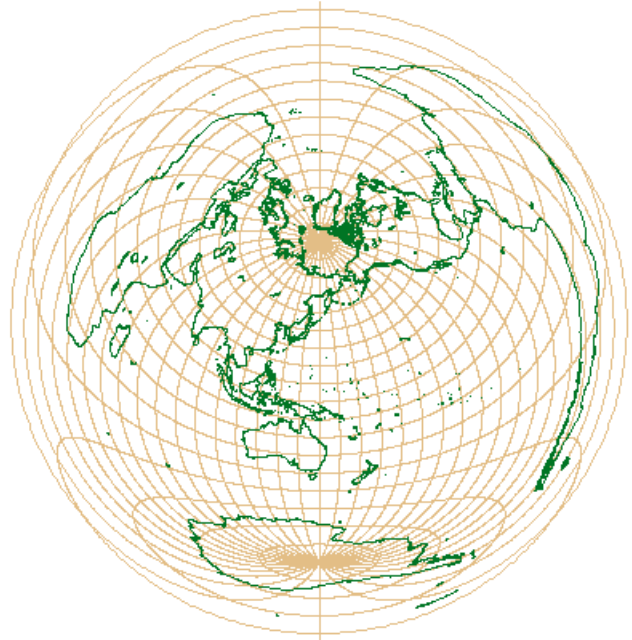
Θ : the polar coordinate east of south

ρ : 투영면의 중심에서의 거리

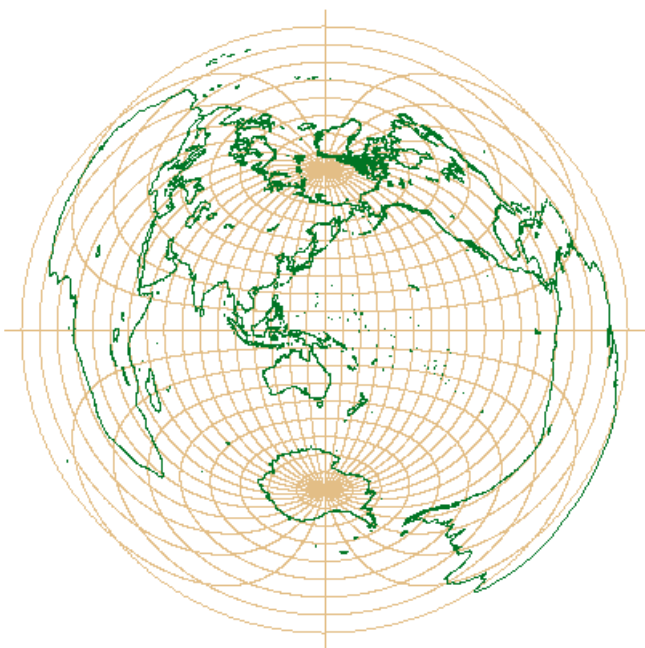
표준위도 = 90N



표준위도 = 45N



표준위도 = 0N



Ⅲ. 지도 변환 실무

1. 지도 변환의 실무 과정

가. 투영법 사용과정

지구상의 표면을 2차원 지도로 옮기기 위해서는 다음의 과정을 거쳐야 한다.

- ① 사용하고자 하는 지구타원체의 사양을 결정하고, 주요 분석영역을 감안하여 지구본의 사양을 결정한다.
 - 지구 반경, 편평도, 주요 대상 위경도 등
- ② 사용할 투영법을 결정한다.
- ③ 투영법의 사양을 결정한다.
 - 점점의 위경도 또는 점선/분할선의 위도
 - 기준 위경도선
 - 지도상의 좌표값을 알고 있는 기준점
- ④ 투영할 지도좌표의 특성을 결정한다.
 - 지도상 거리의 단위 (예, 1 grid = 4 km)
 - 표출할 지도의 영역
 - 위경도를 알고 있는 지도상의 자료값 (좌표변환시 기준이 됨)

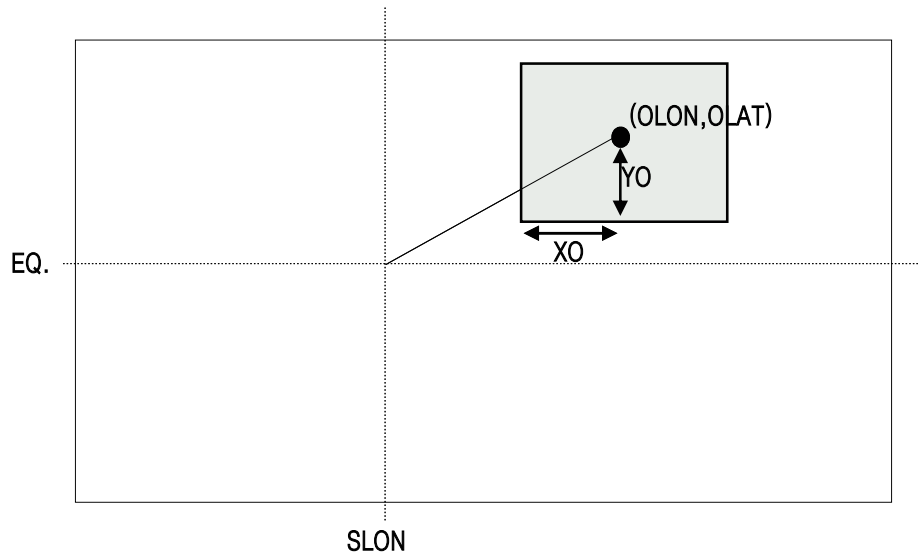
나. 원통도법

● 필요변수 : 투영면의 회전축이 지축과 일치하는 경우

- R : 지구반경 (km)
- SLAT : 투영면이 접하거나 통과하는 위도 (degree)
- SLON : 기준 경도 (degree)
- (OLON,OLAT) : 지도상에 좌표값을 알고 있는 점의 경도,위도 (degree)
- (XO,YO) : (OLON,OLAT)점의 지도상에서의 좌표값 (grid)
- DD : 지도상에서의 거리단위 (km)

● 필요변수 : 투영면의 회전축이 지축과 일치하지 않는 경우(Oblique)

- R : 지구반경 (km)
- (PLON,PLAT) : 투영면의 회전축에서 북극에 해당하는 지점의 경도,위도 (degree)
- (OLON,OLAT) : 지도상에 좌표값을 알고 있는 점의 경도,위도 (degree)
- (XO,YO) : (OLON,OLAT)점의 지도상에서의 좌표값 (grid)
- DD : 지도상에서의 거리단위 (km)

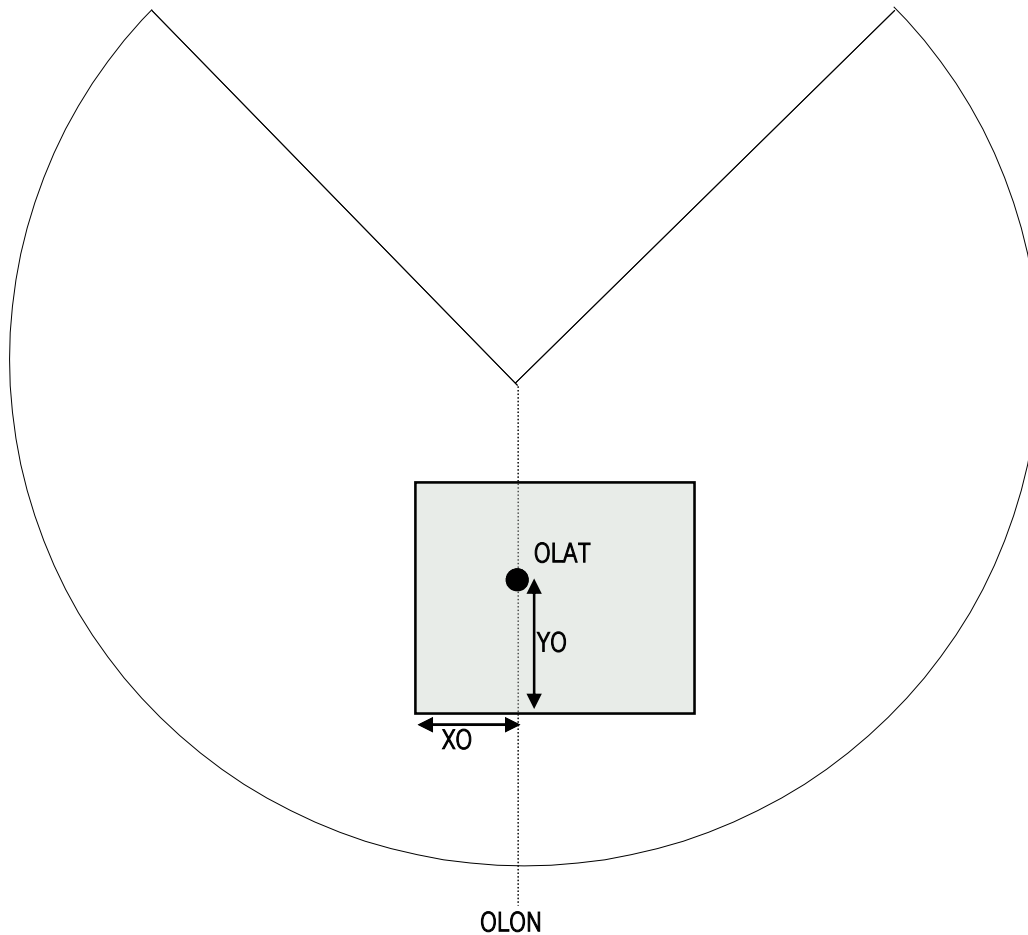


- ▶ 변환 프로그램 예 (Fortran)
 - Mercator projection
 - Transveres Mercator projection
 - Oblique Mercator projection
 - Miller cylindrical projection
 - Equidistant Cylindrical projection
 - Sinusoidal projection
 - Van Der Grinten projection
- ▶ 변환 프로그램 예 (C)
 - Mercator projection
 - Transveres Mercator projection
 - Oblique Mercator projection
 - Miller cylindrical projection
 - Equidistant Cylindrical projection
 - Sinusoidal projection
 - Van Der Grinten projection

다. 원추도법의 경우, 필요한 변수

● 필요변수

- R : 지구반경 (km)
- SLAT1, SLAT2 : 투영면이 접하거나 통과하는 위도 (degree)
(한 위도에서 접하는 경우는 SLAT1 = SLAT2)
- (OLON, OLAT) : 지도상에 좌표값을 알고 있는 점의 경도, 위도 (degree)



* OLON의 경도선은 지도좌표의 y축과 평행함.

- (XO,YO) : (OLON,OLAT)점의 지도상에서의 좌표값 (grid)
- DD : 지도상에서의 거리단위 (km)

▶ 변환 프로그램 예 (Fortran)

- Lambert Conformal Conic projection
- Lambert Equal_Area Conic projection
- Albers Equal-Area Conic projection

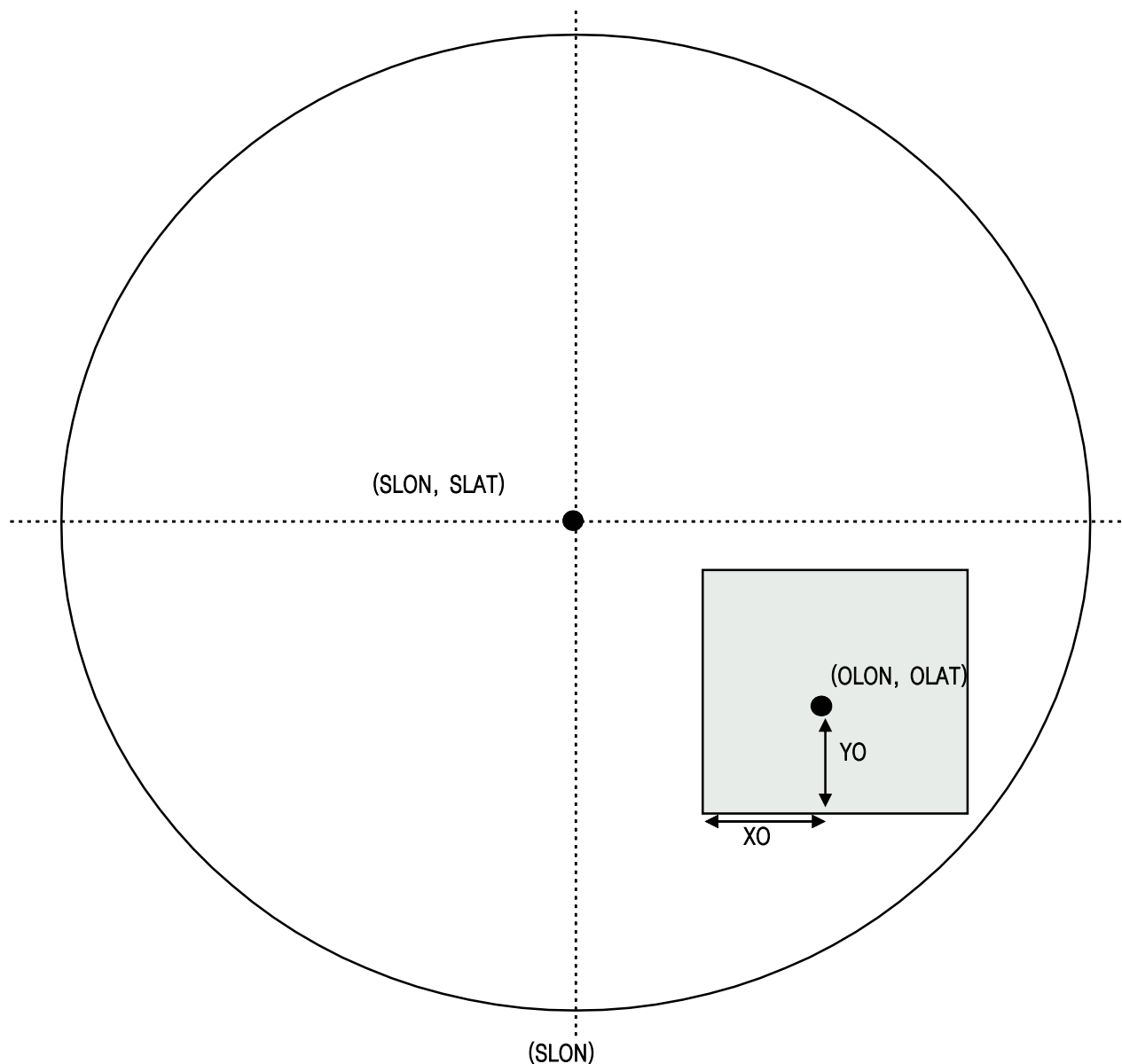
▶ 변환 프로그램 예 (C)

- Lambert Conformal Conic projection
- Lambert Equal_Area Conic projection
- Albers Equal-Area Conic projection

라. 방위도법의 경우

● 필요변수

- R : 지구반경 (km)
- (SLON, SLAT) : 원판의 중심이 접하는 곳의 (degree)
- (OLON, OLAT) : 지도상에 좌표값을 알고 있는 점의 경도, 위도 (degree)
- (XO, YO) : (OLON, OLAT)점의 지도상에서의 좌표값 (grid)
- DD : 지도상에서의 거리단위 (km)



▶ 변환 프로그램 예 (Fortran)

- Stereographic projection
- Orthographic projection
- Azimuthal equidistant projection

- ▶ 변환 프로그램 예 (C)
 - Stereographic projection
 - Orthographic projection
 - Azimuthal equidistant projection

2. 기상청에서 사용중인 지도

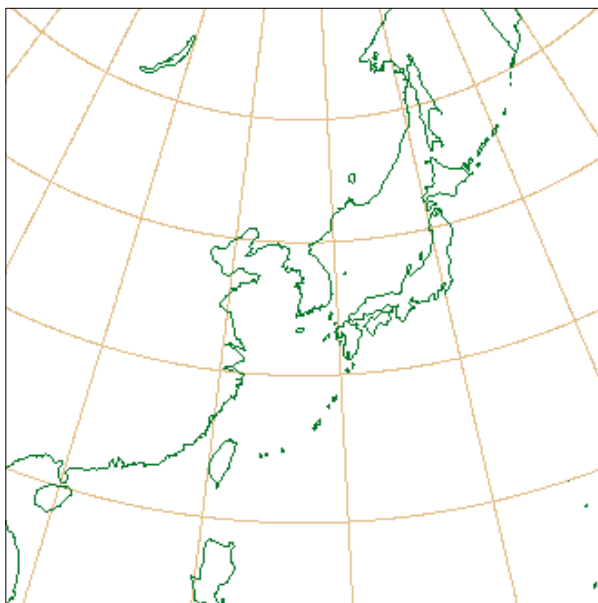
가. 수치일기도

- GDAPS로 생성되는 일기도 (12시간 간격의 지상 및 고층 분석일기도)
 - ▶ 투 영 법 : Lambert Conformal Coonic Projection
 - ▶ 지구반경 : 6371 km
 - ▶ 중 심 점 : 35 N, 125 E
- MM5로 생성되는 일기도 (3시간 간격의 한반도 및 지상3 분석일기도)
 - ▶ 투 영 법 : Lambert Conformal Coonic Projection
 - ▶ 지구반경 : 6370 km
 - ▶ 중 심 점 : 38 N, 126 E
(MM4의 경우 35 N, 125 E)

*) 기타 사양은 표출크기와 일기도 양식에 따라 약간씩 다름

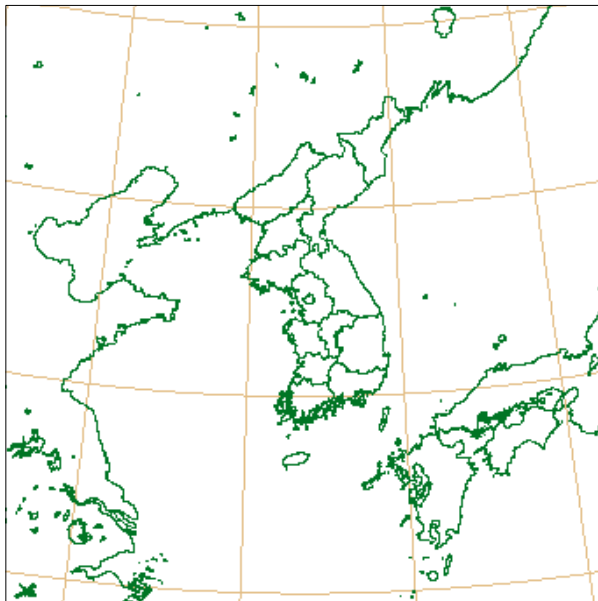
나. GMS 위성

- 아시아영역 자료 (Region)



- ▶ 투 영 법 : 60N을 지나는
Polar Stereographic Projection
- ▶ 지구반경 : 6370.18604 km
- ▶ 중 심 점 : 35N, 127E
- ▶ 표출영역 : 5939.2 km × 5939.2 km
- ▶ 격자간격 : 5.78 km
- ▶ 격 자 수 : 1024 × 1024
- ▶ 사용 타원체 : Clacke(1866)
 - 이심율 1/295
 - 적도반경 6378.206 km

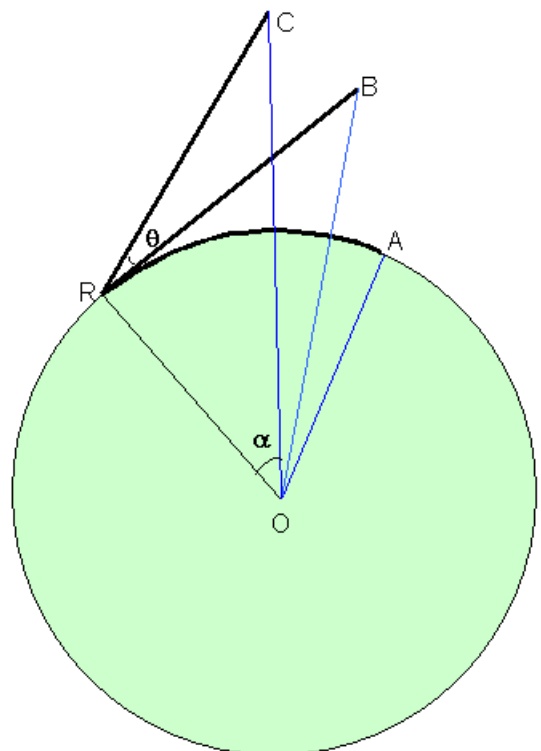
㉠ 한반도영역 자료 (Local)



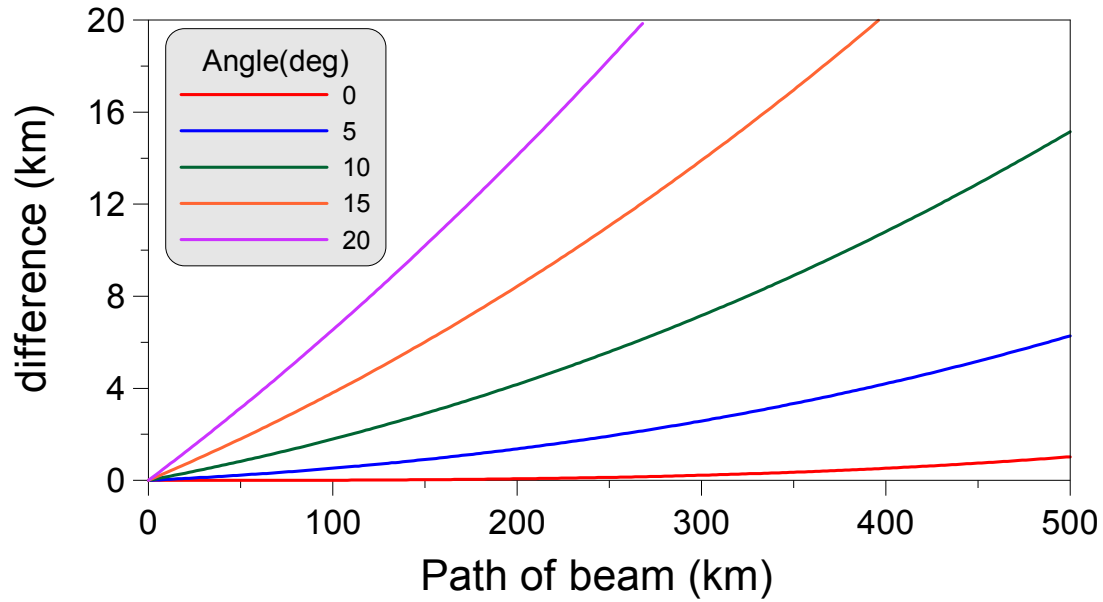
- ▶ 투영법 : 60N을 지나는
Polar Stereographic Projection
- ▶ 지구반경 : 6370.18604 km
- ▶ 중심점 : 37.45N, 126.83E
- ▶ 표출영역 : 2048 km × 2048 km
- ▶ 격자간격 : 2.0 km (또는 4.0 km)
- ▶ 격자수 : 1024 × 1024 (또는 512×512)
- ▶ 사용 타원체 : Clacke(1866)
 - 이심율 1/295
 - 적도반경 6378.206 km

다. 레이더자료

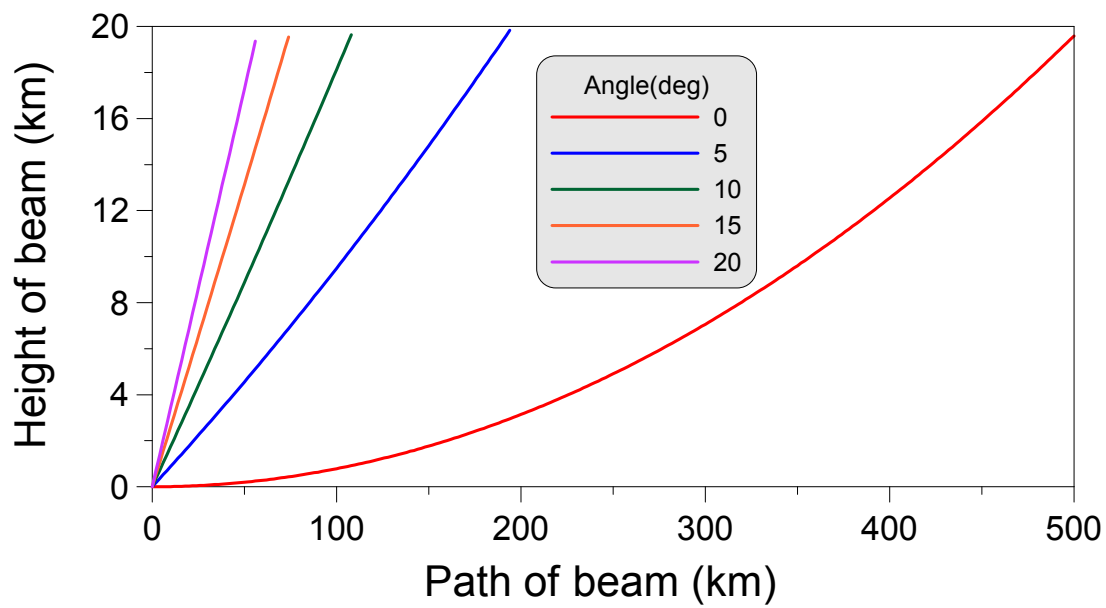
- 레이더의 원시자료는 일반적인 지도투영법을 따르지 않고, 단지 레이더를 중심으로 원형으로 회전하면서 발사하는 빔의 경로에 따라 기하학적으로 결정된다.
- 오른쪽 그림과 같이 일정한 각도로 레이더에서 빔을 발사한 경우에 직선으로 진행한 경우와 지구표면을 따라 진행한 경우의 거리의 차를 빔의 진행거리에 따라 계산하면 [그림 Ⅲ-1]과 같다. [그림 Ⅲ-2]는 빔이 직진한 경우에 지표면에서의 높이를 나타낸 것이다.
이 그림들에서 보면 구름의 높이가 15km 이하라고 할 때, 대부분의 경우에 직진한 경우와 지표면을 따라 진행한 경우의 거리 차는 1km 이하이다.
실제 레이더빔은 대기상태에 따라 지구쪽으로 굴절된 형태의 다양한 곡선경로로 진행하므로, 빔의 Pixel 크기가 수백m 이상인 경우 이 차이는 무시할 수 있다고 생각된다.
- 따라서 레이더자료에 지도투영법을 적용하고자 하면 어떤 기준점에서부터의 거리가 항상 일정한 [정거방위도법\(Azimuthal Equidistant Projection\)](#)을 사용하는 것이 타당하다.



[그림 Ⅲ-1] 각 고도각으로 레이더빔이 직진한 경우와 지표면을 따라 진행한 경우의 거리차

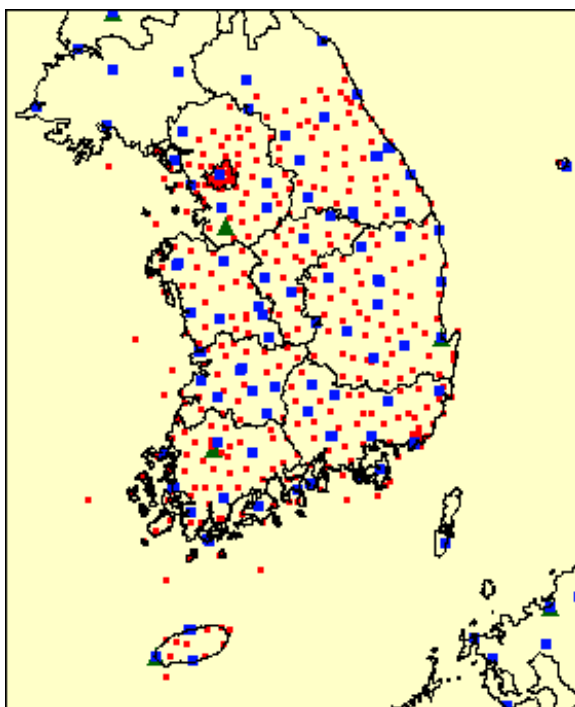


[그림 Ⅲ-2] 어떤 고도각으로 직진한 레이더빔의 지표면에서의 높이



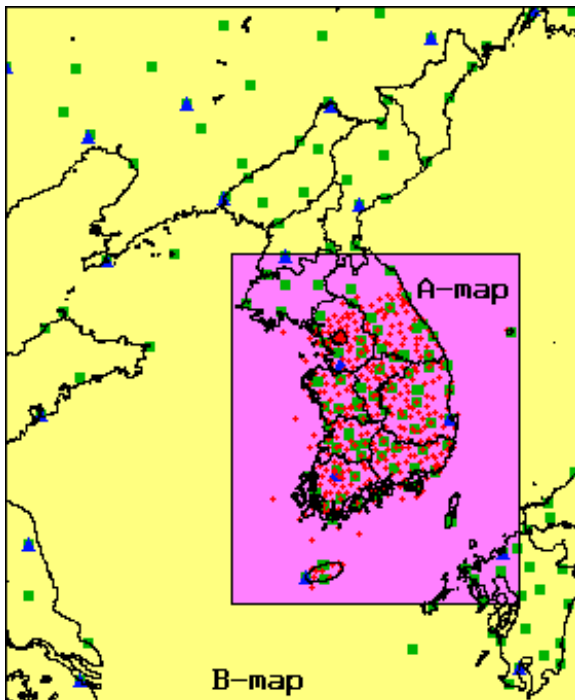
라. 국지기상 연속감시 시스템 분석영역

● A-map



- ▶ 투영법 : Lambert Conformal Conic Projection
- ▶ 기준위도 : 30 N, 60 N
- ▶ 지구반경 : 6370.1846 km
- ▶ 표출영역 : 560 km X 680 km
- ▶ 기준점 : 35 N, 125 E
- ▶ 기준점의 위치 : 서쪽에서 40 km,
동쪽에서 520 km,
남쪽에서 240 km,
북쪽에서 440 km
- ▶ 기본격자간격 : 4 km
- ▶ 격자점수 : 141 X 171

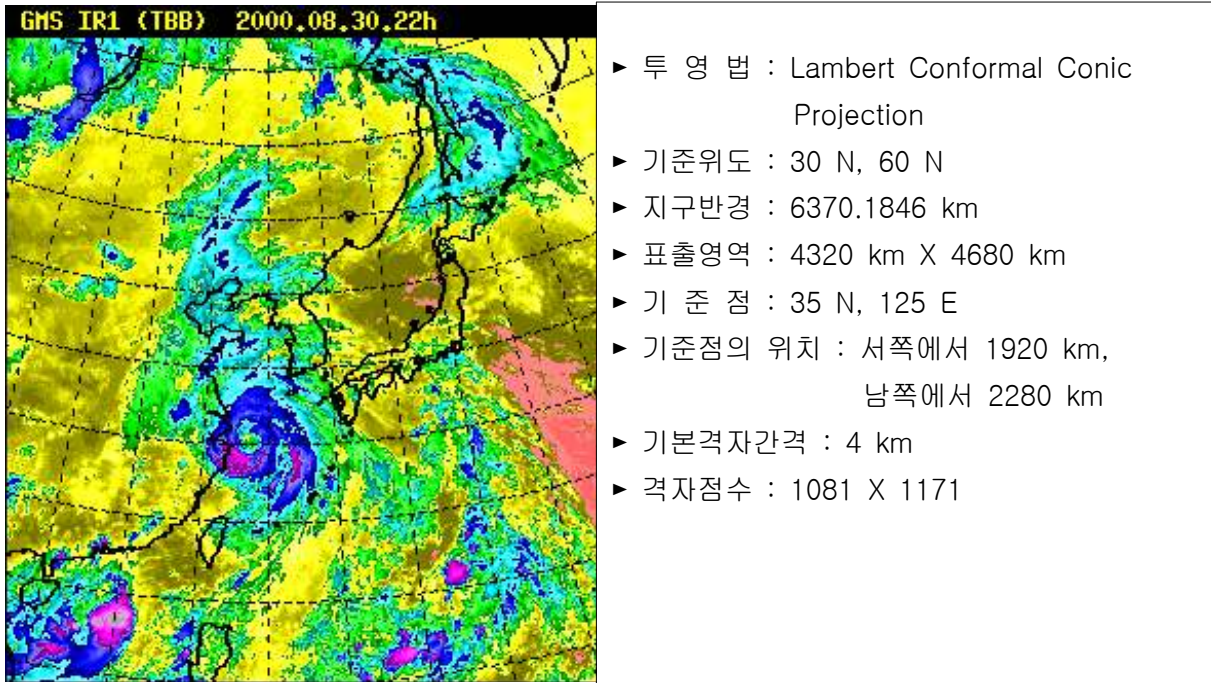
● B-map



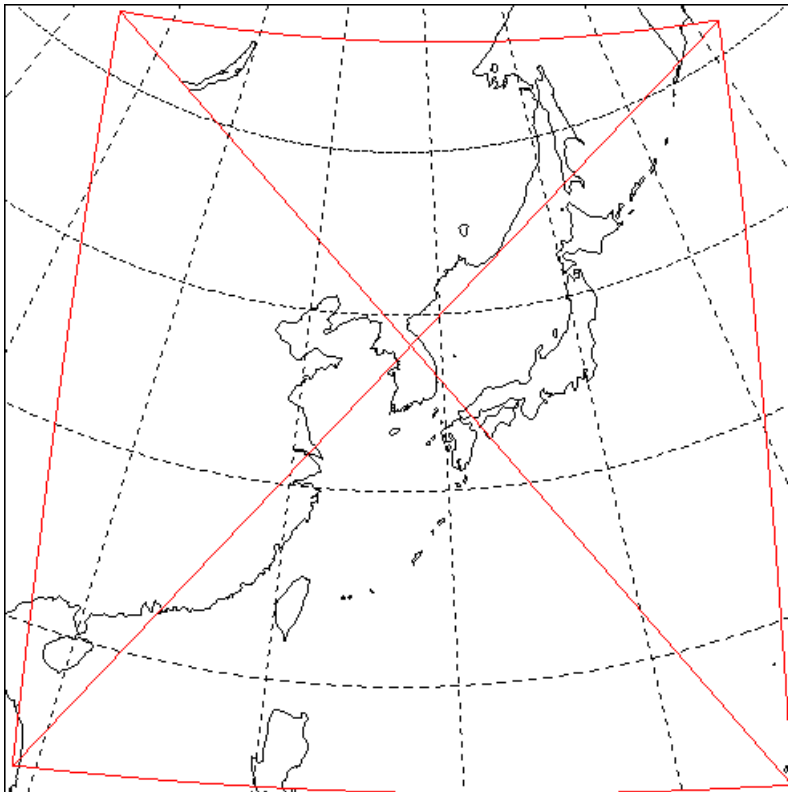
- ▶ 투영법 : Lambert Conformal Conic Projection
- ▶ 기준위도 : 30 N, 60 N
- ▶ 지구반경 : 6370.1846 km
- ▶ 표출영역 : 1120 km X 1360 km
- ▶ 기준점 : 35 N, 125 E
- ▶ 기준점의 위치 : 서쪽에서 480 km,
동쪽에서 640 km,
남쪽에서 440 km,
북쪽에서 920 km
- ▶ 기본격자간격 : 4 km
- ▶ 격자점수 : 281 X 341

마. GMS 자료의 지도 변환

- ㉠ 국지기상연속감시시스템에서는 위성자료를 다른 레이더, 낙뢰, AWS 등과 비교분석할 수 있기 위하여 아래와 같이 위성자료를 지도변환하여 사용하고 있다.



- ㉡ 위와 같이 변환한 영역은 변환전의 위성영역안에 아래와 같이 포함된다.



- 변환영역에서 보듯이 간단한 선형관계식으로는 제대로 변환하기 힘들다. 따라서 Lambert Conformal 지도영역에서의 각 격자점의 (X, Y)좌표를 Lambert Conformal 투영법을 이용하여 위경도로 변환한 다음 Polar Stereographic 투영법으로 원래의 위성자료에서의 위치(x1, y1)를 계산하여 그 지점의 주위 위성값에서 선형내삽하여 하였다.

IV. 지도 변환 프로그램 예

1. Fortran 버전

가. Mercator Projection

```
C*****
      SUBROUTINE MERCPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Mercator projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y       : (x,y) coordinate in map [grid]
C      * N = 0    : (lat,lon) --> (x,y)
C      * N = 1    : (x,y) --> (lat,lon)
C-----
      COMMON /MERCAT/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... ERROR CHECK.
C
      IF (N.EQ.0.AND.ABS(ALAT).GT.89.9) THEN
        PRINT *, 'OVER [ MERCPROJ ]'
        PRINT *, '(LAT,LON) : ',ALAT,ALON
        IERR = 1
        RETURN
      ENDIF
C
C... INITIALIZE.
C
      IF (IFIRST.EQ.0) THEN
        CALL MERCINIT
        XN = OLON - SLON
        IF (XN.GT.PI) XN = XN - 2.*PI
        IF (XN.LT.-PI) XN = 2.*PI + XN
        XO = R*XN - XO
        YO = R*ALOG(TAN(PI*0.25+OLAT*0.5)) - YO
        IFIRST = 1
      ENDIF
C
C... CONVERT.
C
      IF (N.EQ.0) THEN
        XN = ALON*DEGRAD - SLON
        IF (XN.GT.PI) XN = XN - 2.*PI
        IF (XN.LT.-PI) XN = 2.*PI + XN
        X = R*XN - XO
        Y = R*ALOG(TAN(PI*0.25+ALAT*DEGRAD*0.5)) - YO
      ELSEIF (N.EQ.1) THEN
        ALAT = PI*0.5 - 2.*ATAN(EXP(-(Y+YO)/R))
        ALON = (X+XO)/R + SLON
        ALAT = ALAT*RADDEG
        ALON = ALON*RADDEG
      ENDIF
      IERR = 0
      RETURN
      END
```

```

C*****
C      SUBROUTINE MERCINIT
C*****
C      Mercator projection Initialize
C-----
C      COMMON /MERCAT/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG

      PI = ASIN(1.)*2.
      DEGRAD = PI/180.
      RADDEG = 180./PI

      R = 6370.1846          ! [km]      : Earth Radius
      SLAT = 23.5            ! [degree] : Standard Latitude
      SLON = 125.0           ! [degree] : Standard Longitude
      OLAT = 35.0            ! [degree] : Latitude of known point in map
      OLON = 125.0           ! [degree] : Longitude of known point in map
      XO = 0.0               ! [grid]   : X-coordinate of known point
      YO = 0.0               ! [grid]   : Y-coordinate of known point
      DD = 4.0               ! [km]     : Grid distance in map

      SLAT = SLAT*DEGRAD
      SLON = SLON*DEGRAD
      OLAT = OLAT*DEGRAD
      OLON = OLON*DEGRAD
      R = R*COS(SLAT*DEGRAD)/DD

      RETURN
      END

```

4. Transveres mercator projection

```

C*****
SUBROUTINE TRNMPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C   *** Transveres Mercator projection ***
C
C       ALAT, ALON : (latitude,longitude) at earth [degree]
C       X, Y       : (x,y) coordinate in map [grid]
C       * N = 0    : (lat,lon) --> (x,y)
C       * N = 1    : (x,y) --> (lat,lon)
C-----
COMMON /TRNMEC/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
  IF (IFIRST.EQ.0) THEN
    CALL TRNMINIT
    B = COS(OLAT)*SIN(OLON-SLON)
    IF (ABS(B).GE.0.99999) THEN
      PRINT *, 'ERROR [ TRNMPROJ ]'
      PRINT *, '(SLAT,SLON) : ',SLAT*RADDEG,SLON*RADDEG
      PRINT *, '(OLAT,OLON) : ',OLAT*RADDEG,OLON*RADDEG
      STOP
    ENDIF
    XO = 0.5*R*ALOG((1.+B)/(1.-B))-XO
    IF (ABS(OLAT).GE.PI*0.5) THEN
      YN = OLAT
      IF (ABS(OLON-SLON).GT.PI*0.5) YN = -YN
    ELSEIF (ABS(COS(OLON-SLON)).LE.0.0) THEN
      YN = PI*0.5
      IF (OLAT.GT.0.0) YN = -YN
      IF (OLAT.EQ.0.0) YN = 0.
    ELSE
      YN = ATAN2(TAN(OLAT),COS(OLON-SLON))
    ENDIF
    YO = R*(YN-SLAT) - YO
    IFIRST = 1
  ENDIF
C
C... CONVERT.
C
  IF (N.EQ.0) THEN
    BLAT = ALAT*DEGRAD
    BLON = ALON*DEGRAD - SLON
    B = COS(BLAT)*SIN(BLON)
    IF (ABS(B).GE.0.99999) GOTO 999
    X = 0.5*R*ALOG((1.+B)/(1.-B)) - XO
    IF (ABS(ALAT).GE.90.0) THEN
      YN = PI*0.5
      IF (ALAT.LT.0.0) YN = -YN
    ELSEIF (ABS(COS(BLON)).LE.0.0) THEN
      YN = PI*0.5
      IF (ALAT.LT.0.0) YN = -YN
      IF (ALAT.EQ.0.0) YN = 0.
    ELSE
      YN = ATAN2(TAN(BLAT),COS(BLON))
    ENDIF
    Y = R*(YN-SLAT) - YO
  ELSEIF (N.EQ.1) THEN
    XN = X + XO
    YN = Y + YO
    B = YN/R + SLAT
    ALAT = SIN(B)/COSH(XN/R)
    ALAT = ASIN(ALAT)
    IF (ABS(COS(B)).LE.0.00001) THEN
      THETA = PI*0.5
    
```

```

        IF (XN.LT.0.0) THETA = -THETA
        IF (XN.EQ.0.0) THETA = 0.
    ELSE
        THETA = ATAN2(SINH(XN/R),COS(B))
    ENDIF
    ALON = THETA + SLON
    ALAT = ALAT*RADDEG
    ALON = ALON*RADDEG
ENDIF
RETURN
C
C... ERROR.
C
    999 CONTINUE
    PRINT *, 'OVER [ TRNMPROJ ]'
    PRINT *, '(SLAT,SLON) :',SLAT*RADDEG,SLON*RADDEG
    PRINT *, '(OLAT,OLON) :',OLAT*RADDEG,OLON*RADDEG
    PRINT *, '(ALAT,ALON) :',ALAT,ALON
    IERR = 1
    RETURN
    END

C*****
SUBROUTINE TRNMINIT
C*****
C Transveres Mercator projection Initialize
C
C-----
COMMON /TRNMEC/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
COMMON /CALCUL/ PI,DEGRAD,RADDEG

PI = ASIN(1.)*2.
DEGRAD = PI/180.
RADDEG = 180./PI

R = 6370.1846      ! [km]      : Earth Radius
SLAT = 35.0        ! [degree] : Standard Latitude
SLON = 125.0       ! [degree] : Standard Longitude
OLAT = 35.0        ! [degree] : Latitude of known point in map
OLON = 125.0       ! [degree] : Longitude of known point in map
XO = 0.0           ! [grid]  : X-coordinate of known point
YO = 0.0           ! [grid]  : Y-coordinate of known point
DD = 4.0           ! [km]    : Grid distance in map

SLAT = SLAT*DEGRAD
SLON = SLON*DEGRAD
OLAT = OLAT*DEGRAD
OLON = OLON*DEGRAD
R = R/DD

RETURN
END

```


다. Oblique mercator projection

```

C*****
C      SUBROUTINE OBLMPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Oblique Mercator projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y       : (x,y) coordinate in map [grid]
C      * N = 0    : (lat,lon) --> (x,y)
C      * N = 1    : (x,y) --> (lat,lon)
C-----
C      COMMON /OBLMER/ R,PLAT,PLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C        CALL OBLMINIT
C        B = SIN(OLON-PLON)
C        IF (ABS(OLAT).GE.PI*0.5) THEN
C          XN = PI*0.5
C          IF (OLAT*B.LT.0.0) XN = -XN
C        ELSE
C          C = TAN(OLAT)*COS(PLAT)-SIN(PLAT)*COS(OLON-PLON)
C          IF (ABS(C).LE.0.0) THEN
C            XN = 0.
C          ELSE
C            IF (ABS(B).LE.0.00001) THEN
C              XN = PI*0.5
C              IF (C*B.LT.0.0) XN = -XN
C            ELSE
C              XN = ATAN2(C,B)
C            ENDIF
C          ENDIF
C        ENDIF
C        XO = R*XN - XO
C        A = SIN(PLAT)*SIN(OLAT)+COS(PLAT)*COS(OLAT)*COS(OLON-PLON)
C        IF (ABS(A).GE.0.99999) THEN
C          PRINT *, 'ERROR [ OBLMPROJ ]'
C          PRINT *, '(OLAT,OLON) : ', OLAT*RADDEG, OLON*RADDEG
C          PRINT *, '(ALAT,ALON) : ', ALAT*RADDEG, ALON*RADDEG
C          STOP
C        ENDIF
C        YO = 0.5*R*A*LOG((1.+A)/(1.-A)) - YO
C        IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C        BLAT = ALAT*DEGRAD
C        BLON = ALON*DEGRAD - PLON
C        B = SIN(BLON)
C        IF (ABS(ALAT).GE.90.0) THEN
C          XN = PI*0.5
C          IF (ALAT.LT.0.0) XN = -XN
C        ELSE
C          C = TAN(BLAT)*COS(PLAT)-SIN(PLAT)*COS(BLON)
C          IF (ABS(C).LE.0.0) THEN
C            XN = 0.
C          ELSE
C            IF (ABS(B).LE.0.00001) THEN
C              B1 = COS(BLON)
C              IF ((B1.GT.0.0.AND.BLAT.LT.PLAT).OR.
C                *      (B1.LT.0.0.AND.BLAT.LT.-PLAT)) THEN
C                XN = -PI*0.5
C              ELSE
C                XN = PI*0.5
C              ENDIF
C            ENDIF
C          ENDIF
C        ENDIF
C      ENDIF

```

```

        ELSE
            XN = ATAN2(C,B)
        ENDIF
    ENDIF
    ENDIF
    X = R*XN - XO
    A = SIN(PLAT)*SIN(BLAT)+COS(PLAT)*COS(BLAT)*COS(BLON)
    IF (ABS(A).GE.0.99999) GOTO 999
    Y = 0.5*R*ALOG((1.+A)/(1.-A)) - YO
    ELSEIF (N.EQ.1) THEN
        XN = (X+XO)/R
        YN = (Y+YO)/R
        ALAT = SIN(PLAT)*TANH(YN) + COS(PLAT)*SIN(XN)/COSH(YN)
        ALAT = ASIN(ALAT)
        C = SIN(PLAT)*SIN(XN) - COS(PLAT)*SINH(YN)
        B = COS(XN)
        IF (ABS(C).LE.0.0) THEN
            ALON = 0.
        ELSEIF (ABS(B).LE.0.00001) THEN
            ALON = PI*0.5
            IF (C*B.LT.0.0) ALON = -ALON
        ELSE
            ALON = ATAN2(C,B) + PI*0.5
        ENDIF
        ALON = ALON + PLON
        ALAT = ALAT*RADDEG
        ALON = ALON*RADDEG
    ENDIF
    RETURN
C
C... ERROR.
C
999 CONTINUE
    PRINT *, 'OVER [ OBLMPROJ ]'
    PRINT *, '(PLAT,PLON) : ', PLAT*RADDEG, PLON*RADDEG
    PRINT *, '(OLAT,OLON) : ', OLAT*RADDEG, OLON*RADDEG
    PRINT *, '(ALAT,ALON) : ', ALAT, ALON
    IERR = 1
    RETURN
    END
C*****
    SUBROUTINE OBLMINIT
C*****
C    Oblique Mercator projection Initialize
C
C-----
    COMMON /OBLMER/ R,PLAT,PLON,OLAT,OLON,XO,YO,IFIRST
    COMMON /CALCUL/ PI,DEGRAD,RADDEG

    PI = ASIN(1.)*2.
    DEGRAD = PI/180.
    RADDEG = 180./PI

    R = 6370.1846          ! [km]      : Earth Radius
    PLAT = 23.5            ! [degree] : Latitude of Oblique Polar
    PLON = 125.0           ! [degree] : Longitude of Oblique Polar
    OLAT = 35.0            ! [degree] : Latitude of known point in map
    OLON = 125.0           ! [degree] : Longitude of known point in map
    XO = 0.0               ! [grid]   : X-coordinate of known point
    YO = 0.0               ! [grid]   : Y-coordinate of known point
    DD = 4.0               ! [km]     : Grid distance in map

    PLAT = PLAT*DEGRAD
    PLON = PLON*DEGRAD
    OLAT = OLAT*DEGRAD
    OLON = OLON*DEGRAD
    R = R/DD

    RETURN
    END

```

라. Miller cylindrical projection

```

C*****
C      SUBROUTINE MILMPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Miller Cylindrical projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y       : (x,y) coordinate in map [grid]
C      * N = 0    : (lat,lon) --> (x,y)
C      * N = 1    : (x,y) --> (lat,lon)
C-----
C      COMMON /MILLCY/ R,SLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C          CALL MILMINIT
C          XN = OLON - SLON
C          IF (XN.GT.PI) XN = XN - 2.*PI
C          IF (XN.LT.-PI) XN = 2.*PI + XN
C          XO = R*XN - XO
C          YO = R*ALOG(TAN(PI*0.25+OLAT*0.4))/0.8 - YO
C          IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C          XN = ALON*DEGRAD - SLON
C          IF (XN.GT.PI) XN = XN - 2.*PI
C          IF (XN.LT.-PI) XN = 2.*PI + XN
C          X = R*XN - XO
C          Y = R*ALOG(TAN(PI*0.25+ALAT*DEGRAD*0.4))/0.8 - YO
C      ELSEIF (N.EQ.1) THEN
C          ALAT = 2.5*ATAN(EXP(0.8*(Y+YO)/R)) - 5.*PI/8.
C          ALON = (X+XO)/R + SLON
C          ALAT = ALAT*RADDEG
C          ALON = ALON*RADDEG
C      ENDIF
C      IERR = 0
C      RETURN
C      END
C*****
C      SUBROUTINE MILMINIT
C*****
C      Miller Cylindrical projection Initialize
C-----
C      COMMON /MILLCY/ R,SLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C      PI = ASIN(1.)*2.
C      DEGRAD = PI/180.
C      RADDEG = 180./PI
C
C      R = 6370.1846      ! [km]      : Earth Radius
C      SLON = 125.0       ! [degree] : Standard Longitude
C      OLAT = 35.0        ! [degree] : Latitude of known point in map
C      OLON = 125.0       ! [degree] : Longitude of known point in map
C      XO = 0.0           ! [grid]   : X-coordinate of known point
C      YO = 0.0           ! [grid]   : Y-coordinate of known point
C      DD = 4.0           ! [km]     : Grid distance in map
C
C      SLON = SLON*DEGRAD
C      OLAT = OLAT*DEGRAD
C      OLON = OLON*DEGRAD
C      R = R/DD
C
C      RETURN
C      END

```

04. Equidistant cylindrical projection

```

C*****
SUBROUTINE EQDCPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C *** Equidistant Cylindrical projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y      : (x,y) coordinate in map [grid]
C      * N = 0   : (lat,lon) --> (x,y)
C      * N = 1   : (x,y) --> (lat,lon)
C-----
COMMON /EQUDIS/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C          CALL EQDCINIT
C          XN = OLON - SLON
C          IF (XN.GT.PI) XN = XN - 2.*PI
C          IF (XN.LT.-PI) XN = 2.*PI + XN
C          XO = R*XN*COS(SLAT) - XO
C          IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C          XN = ALON*DEGRAD - SLON
C          IF (XN.GT.PI) XN = XN - 2.*PI
C          IF (XN.LT.-PI) XN = 2.*PI + XN
C          X = R*XN*COS(SLAT) - XO
C          Y = R*(ALAT*DEGRAD-OLAT) + YO
C      ELSEIF (N.EQ.1) THEN
C          ALAT = (Y-YO)/R + OLAT
C          ALON = (X+XO)/(R*COS(SLAT)) + SLON
C          ALAT = ALAT*RADDEG
C          ALON = ALON*RADDEG
C      ENDIF
C      IERR = 0
C      RETURN
C      END
C*****
SUBROUTINE EQDCINIT
C*****
C Equidistant Cylindrical projection Initialize
C-----
COMMON /EQUDIS/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
COMMON /CALCUL/ PI,DEGRAD,RADDEG

PI = ASIN(1.)*2.
DEGRAD = PI/180.
RADDEG = 180./PI

R = 6370.1846      ! [km]      : Earth Radius
SLAT = 0.0         ! [degree] : Standard Latitude
SLON = 125.0       ! [degree] : Standard Longitude
OLAT = 35.0        ! [degree] : Latitude of known point in map
OLON = 125.0       ! [degree] : Longitude of known point in map
XO = 0.0           ! [grid]   : X-coordinate of known point
YO = 0.0           ! [grid]   : Y-coordinate of known point
DD = 4.0           ! [km]     : Grid distance in map

SLAT = SLAT*DEGRAD
SLON = SLON*DEGRAD
OLAT = OLAT*DEGRAD
OLON = OLON*DEGRAD
R = R/DD

RETURN
END

```

바. Sinusoidal projection

```

C*****
C      SUBROUTINE SINUPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Sinusoidal projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y      : (x,y) coordinate in map [grid]
C      * N = 0   : (lat,lon) --> (x,y)
C      * N = 1   : (x,y) --> (lat,lon)
C-----
C      COMMON /SINUSO/ R,SLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C          CALL SINUINIT
C          XO = R*(OLON-SLON)*COS(OLAT) - XO
C          IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C          BLAT = ALAT*DEGRAD
C          BLON = ALON*DEGRAD - SLON
C          X = R*(BLON)*COS(BLAT) - XO
C          Y = R*(BLAT-OLAT) + YO
C      ELSEIF (N.EQ.1) THEN
C          ALAT = (Y-YO)/R + OLAT
C          ALON = (X+XO)/(R*COS(ALAT)) + SLON
C          ALAT = ALAT*RADDEG
C          ALON = ALON*RADDEG
C      ENDIF
C      IERR = 0
C      RETURN
C      END
C*****
C      SUBROUTINE SINUINIT
C*****
C      Sinusoidal projection Initialize
C-----
C      COMMON /SINUSO/ R,SLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C      PI = ASIN(1.)*2.
C      DEGRAD = PI/180.
C      RADDEG = 180./PI
C
C      R = 6370.1846      ! [km]      : Earth Radius
C      SLON = 125.0      ! [degree] : Standard Longitude
C      OLAT = 35.0       ! [degree] : Latitude of known point in map
C      OLON = 125.0      ! [degree] : Longitude of known point in map
C      XO = 0.0          ! [grid]   : X-coordinate of known point
C      YO = 0.0          ! [grid]   : Y-coordinate of known point
C      DD = 4.0          ! [km]    : Grid distance in map
C
C      SLON = SLON*DEGRAD
C      OLAT = OLAT*DEGRAD
C      OLON = OLON*DEGRAD
C      R = R/DD
C
C      RETURN
C      END

```

사. Van der grinten projection

```

C*****
C      SUBROUTINE VANDPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Van der grinten projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y      : (x,y) coordinate in map [grid]
C      * N = 0   : (lat,lon) --> (x,y)
C      * N = 1   : (x,y) --> (lat,lon)
C-----
C      COMMON /VANDER/ R,SLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C        CALL VANDINIT
C        IF (OLAT.EQ.0.0) THEN
C          XN = R*ABS(OLON-SLON)
C          YN = 0
C        ELSEIF (OLON.EQ.SLON) THEN
C          THETA = ASIN(ABS(2.*OLAT/PI))
C          XN = 0
C          YN = PI*R*TAN(THETA*0.5)
C        ELSE
C          THETA = ASIN(ABS(2.*OLAT/PI))
C          A = 0.5*ABS(PI/(OLON-SLON)-(OLON-SLON)/PI)
C          G = COS(THETA)/(SIN(THETA)+COS(THETA)-1.)
C          P = G*(2./COS(THETA)-1.)
C          XN = A*(G-P*P)+SQRT(A*A*(G-P*P)**2-(P*P+A*A)*(G*G-P*P))
C          XN = PI*R*XN/(A*A+P*P)
C          YN = PI*R*SQRT(1.-(XN/(PI*R))**2-2.*A*ABS(XN/(PI*R)))
C        ENDIF
C        IF (OLON.LT.SLON) XN = -XN
C        IF (OLAT.LT.0.0) YN = -YN
C        XO = XN - XO
C        YO = YN - YO
C        IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C        BLAT = ALAT*DEGRAD
C        BLON = ALON*DEGRAD - SLON
C        IF (BLAT.EQ.0.0) THEN
C          X = R*ABS(BLON)
C          Y = 0
C        ELSEIF (ABS(BLON).EQ.0.0) THEN
C          X = 0
C          THETA = ASIN(ABS(2.*BLAT/PI))
C          Y = PI*R*TAN(THETA*0.5)
C        ELSE
C          THETA = ASIN(ABS(2.*BLAT/PI))
C          A = 0.5*ABS(PI/BLON-BLON/PI)
C          G = COS(THETA)/(SIN(THETA)+COS(THETA)-1.)
C          P = G*(2./SIN(THETA)-1.)
C          X = A*(G-P*P)+SQRT(A*A*(G-P*P)**2-(P*P+A*A)*(G*G-P*P))
C          X = PI*R*X/(A*A+P*P)
C          Y = 1.-(X/(PI*R))**2-2.*A*ABS(X/(PI*R))
C          IF (Y.LE.0.0) THEN
C            Y = 0.
C          ELSE
C            Y = PI*R*SQRT(Y)
C          ENDIF
C        ENDIF
C      ENDIF
C      IF (BLON.LT.0.0) X = -X
C      IF (BLAT.LT.0.0) Y = -Y

```

```

      X = X - XO
      Y = Y - YO
    ELSEIF (N.EQ.1) THEN
      XN = X + XO
      YN = Y + YO
      IF (XN.EQ.0.) THEN
        THETA = 2.*ATAN(YN/(PI*R))
        ALAT = 0.5*PI*SIN(THETA)
        ALON = OLON
      ELSEIF (YN.EQ.0.) THEN
        ALAT = 0.
        ALON = XN/R + OLON
      ELSE
        IERR = 1
      ENDIF
      ALAT = ALAT*RADDEG
      ALON = ALON*RADDEG
    ENDIF
    RETURN
  END

```

```

C*****
SUBROUTINE VANDINIT

```

```

C*****

```

```

C  Van der grinten projection Initialize

```

```

C

```

```

C-----

```

```

      COMMON /VANDER/ R,SLON,OLAT,OLON,XO,YO,IFIRST
      COMMON /CALCUL/ PI,DEGRAD,RADDEG

```

```

      PI = ASIN(1.)*2.
      DEGRAD = PI/180.
      RADDEG = 180./PI

```

```

      R = 6370.1846      ! [km]      : Earth Radius
      SLON = 125.0       ! [degree] : Standard Longitude
      OLAT = 35.0        ! [degree] : Latitude of known point in map
      OLON = 125.0       ! [degree] : Longitude of known point in map
      XO = 0.0           ! [grid]   : X-coordinate of known point
      YO = 0.0           ! [grid]   : Y-coordinate of known point
      DD = 4.0           ! [km]     : Grid distance in map

```

```

      SLON = SLON*DEGRAD
      OLAT = OLAT*DEGRAD
      OLON = OLON*DEGRAD
      R = R/DD

```

```

      RETURN
    END

```

04. Lambert conformal conic projection

```

C*****
C      SUBROUTINE LAMC PROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Lambert Conformal Conic Projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y      : (x,y) coordinate in map [grid]
C      * N = 0   : (lat,lon) --> (x,y)
C      * N = 1   : (x,y) --> (lat,lon)
C-----
C      COMMON /LAMCON/ R,SLAT1,SLAT2,SN,SF,RO,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C        CALL LAMCINIT
C        IF (SLAT1.EQ.SLAT2.OR.
1      ABS(SLAT1).GE.PI*0.5.OR.ABS(SLAT2).GE.PI*0.5) THEN
C          PRINT *, 'ERROR [ LAMC PROJ ]'
C          PRINT *, '(SLAT1,SLAT2) : ',SLAT1*RADDEG,SLAT2*RADDEG
C          STOP
C        ENDIF
C        SN = TAN(PI*0.25+SLAT2*0.5)/TAN(PI*0.25+SLAT1*0.5)
C        SN = ALOG(COS(SLAT1)/COS(SLAT2))/ALOG(SN)
C        SF = (TAN(PI*0.25+SLAT1*0.5))*SN*COS(SLAT1)/SN
C        IF (ABS(OLAT).GT.89.9*DEGRAD) THEN
C          IF (SN*OLAT.LT.0.0) THEN
C            PRINT *, 'ERROR [ LAMC PROJ ]'
C            PRINT *, '(SLAT1,SLAT2) : ',SLAT1*RADDEG,SLAT2*RADDEG
C            PRINT *, '(OLAT ,OLON ) : ',OLAT*RADDEG,OLON*RADDEG
C            STOP
C          ENDIF
C          RO = 0.
C        ELSE
C          RO = R*SF/(TAN(PI*0.25+OLAT*0.5))*SN
C        ENDIF
C        IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C        IF (ABS(ALAT).GT.89.9) THEN
C          IF (SN*ALAT.LT.0.0) GOTO 999
C          RA = 0.
C        ELSE
C          RA = R*SF/(TAN(PI*0.25+ALAT*DEGRAD*0.5))*SN
C        ENDIF
C        THETA = ALON*DEGRAD - OLON
C        IF (THETA.GT.PI) THETA = THETA - 2.*PI
C        IF (THETA.LT.-PI) THETA = 2.*PI + THETA
C        THETA = SN*THETA
C        X = RA*SIN(THETA) + XO
C        Y = RO - RA*COS(THETA) + YO
C      ELSEIF (N.EQ.1) THEN
C        XN = X - XO
C        YN = RO - Y + YO
C        RA = SQRT(XN*XN+YN*YN)
C        IF (SN.LT.0) RA = -RA
C        ALAT = (R*SF/RA)**(1./SN)
C        ALAT = 2.*ATAN(ALAT)-PI*0.5
C        IF (ABS(XN).LE.0.0) THEN
C          THETA = 0.
C        ELSE
C          IF (ABS(YN).LE.0.0) THEN
C            THETA = PI*0.5

```



```

        IF (XN.LT.0.0) THETA = -THETA
      ELSE
        THETA = ATAN2(XN,YN)
      ENDIF
    ENDIF
    ALON = THETA/SN + OLO
    ALAT = ALAT*RADDEG
    ALON = ALON*RADDEG
  ENDIF
  RETURN
C
C... ERROR.
C
  999 CONTINUE
  PRINT *, 'OVER [ LAMC PROJ ]'
  PRINT *, '(SLAT1,SLAT2) : ',SLAT1*RADDEG,SLAT2*RADDEG
  PRINT *, '( OLAT, OLO ) : ',OLAT*RADDEG,OLO*RADDEG
  PRINT *, '( ALAT, ALON ) : ',ALAT,ALON
  IERR = 1
  RETURN
END

C*****
SUBROUTINE LAMCINIT
C*****
C Lambert Conformal Conic Projection Initialize
C
C-----
COMMON /LAMCON/ R,SLAT1,SLAT2,SN,SF,RO,OLAT,OLON,XO,YO,IFIRST
COMMON /CALCUL/ PI,DEGRAD,RADDEG

PI = ASIN(1.)*2.
DEGRAD = PI/180.
RADDEG = 180./PI

R = 6370.19584      ! [km]      : Earth Radius
SLAT1 = 30.0        ! [degree] : Standard Latitude 1
SLAT2 = 60.0        ! [degree] : Standard Latitude 2
OLAT = 35.0         ! [degree] : Latitude of known point in map
OLON = 125.0        ! [degree] : Longitude of known point in map
XO = 120.0          ! [grid]   : X-coordinate of known point
YO = 110.0          ! [grid]   : Y-coordinate of known point
DD = 4.0            ! [km]     : Grid distance in map

SLAT1 = SLAT1*DEGRAD
SLAT2 = SLAT2*DEGRAD
OLAT = OLAT*DEGRAD
OLON = OLON*DEGRAD
R = R/DD

RETURN
END

```

자. Lambert azimuthal equal-area projection

```

C*****
C      SUBROUTINE LAMEPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Lambert azimuthal equal-area projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y       : (x,y) coordinate in map [grid]
C      * N = 0    : (lat,lon) --> (x,y)
C      * N = 1    : (x,y) --> (lat,lon)
C-----
C      COMMON /LAMBAZ/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C        CALL LAMEINIT
C        AK = 1./SIN(SLAT)*SIN(OLAT)+COS(SLAT)*COS(OLAT)*COS(OLON-SLON)
C        IF (AK.LE.0.00001) THEN
C          XO = -XO
C          YO = -YO
C        ELSE
C          AK = SQRT(2./AK)
C          XO = R*AK*COS(OLAT)*SIN(OLON-SLON) - XO
C          YO = R*AK*(COS(SLAT)*SIN(OLAT)
C          *      -SIN(SLAT)*COS(OLAT)*COS(OLON-SLON)) - YO
C        ENDIF
C        IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C        BLAT = ALAT*DEGRAD
C        BLON = ALON*DEGRAD - SLON
C        AK = 1./SIN(SLAT)*SIN(BLAT)+COS(SLAT)*COS(BLAT)*COS(BLON)
C        IF (AK.LE.0.00001) GOTO 999
C        AK = SQRT(2./AK)
C        X = R*AK*COS(BLAT)*SIN(BLON) - XO
C        Y = R*AK*(COS(SLAT)*SIN(BLAT)
C        *      -SIN(SLAT)*COS(BLAT)*COS(BLON)) - YO
C      ELSEIF (N.EQ.1) THEN
C        XN = X + XO
C        YN = Y + YO
C        RA = XN*XN + YN*YN
C        IF (RA.LE.0.0) THEN
C          AC = 0.
C          ALAT = SLAT
C        ELSE
C          RA = SQRT(RA)
C          AC = 2.*ASIN(0.5*RA/R)
C          ALAT = COS(AC)*SIN(SLAT) + YN*SIN(AC)*COS(SLAT)/RA
C          ALAT = ASIN(ALAT)
C        ENDIF
C        A1 = XN*SIN(AC)
C        A2 = RA*COS(SLAT)*COS(AC)-YN*SIN(SLAT)*SIN(AC)
C        IF (ABS(A1).LE.0.0) THEN
C          THETA = 0.
C        ELSEIF (ABS(A2).LE.0.0) THEN
C          THETA = PI*0.5
C          IF (A2.LT.0.0) THETA = -THETA
C        ELSE
C          THETA = ATAN2(A1,A2)
C        ENDIF
C        ALON = THETA + SLON
C        ALAT = ALAT*RADDEG
C        ALON = ALON*RADDEG

```

```

        ENDIF
        RETURN
C
C... ERROR.
C
999 CONTINUE
PRINT *, 'OVER [ LAMEPROJ ]'
PRINT *, '(SLAT,SLON) : ', SLAT*RADDEG, SLON*RADDEG
PRINT *, '(OLAT,OLON) : ', OLAT*RADDEG, OLON*RADDEG
PRINT *, '(ALAT,ALON) : ', ALAT, ALON
IERR = 1
RETURN
END

C*****
SUBROUTINE LAMEINIT
C*****
C Lambert azimuthal equal-area projection Initialize
C-----
COMMON /LAMBAZ/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
COMMON /CALCUL/ PI,DEGRAD,RADDEG

PI = ASIN(1.)*2.
DEGRAD = PI/180.
RADDEG = 180./PI

R = 6370.1846      ! [km]      : Earth Radius
SLAT = 90.0        ! [degree] : Standard Latitude
SLON = 125.0       ! [degree] : Standard Longitude
OLAT = 35.0        ! [degree] : Latitude of known point in map
OLON = 125.0       ! [degree] : Longitude of known point in map
XO = 0.0           ! [grid]  : X-coordinate of known point
YO = 0.0           ! [grid]  : Y-coordinate of known point
DD = 4.0           ! [km]    : Grid distance in map

SLAT = SLAT*DEGRAD
SLON = SLON*DEGRAD
OLAT = OLAT*DEGRAD
OLON = OLON*DEGRAD
R = R/DD

RETURN
END

```

차. albers equal-area conic projection

```

C*****
C      SUBROUTINE ALBEPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Albers Equal-area Conic projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y      : (x,y) coordinate in map [grid]
C      * N = 0   : (lat,lon) --> (x,y)
C      * N = 1   : (x,y) --> (lat,lon)
C-----
C      COMMON /ALBERC/ R,SLAT1,SLAT2,SN,SC,RO,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C        CALL ALBEINIT
C        SN = (SIN(SLAT1)+SIN(SLAT2))*0.5
C        IF (ABS(SN).LE.0.0) THEN
C          PRINT *, 'ERROR [ ALBEPROJ ]'
C          PRINT *, '(SLAT1,SLAT2) : ',SLAT1*RADDEG,SLAT2*RADDEG
C          STOP
C        ENDIF
C        SC = 1.+SIN(SLAT1)*SIN(SLAT2)
C        IF (SN*OLAT.LT.0.0.AND.ABS(OLAT).GT.89.9*DEGRAD) THEN
C          PRINT *, 'ERROR [ ALBEPROJ ]'
C          PRINT *, '(SLAT1,SLAT2) : ',SLAT1*RADDEG,SLAT2*RADDEG
C          PRINT *, '(OLAT ,OLON ) : ',OLAT*RADDEG,OLON*RADDEG
C          STOP
C        ENDIF
C        RO = R*SQRT(SC-2.*SN*SIN(OLAT))/SN
C        IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C        IF (SN*ALAT.LT.0.0.AND.ABS(ALAT).GT.89.9) GOTO 999
C        RA = R*SQRT(SC-2.*SN*SIN(ALAT*DEGRAD))/SN
C        THETA = ALON*DEGRAD - OLON
C        IF (THETA.GT.PI) THETA = THETA - 2.*PI
C        IF (THETA.LT.-PI) THETA = 2.*PI + THETA
C        THETA = SN*THETA
C        X = RA*SIN(THETA) + XO
C        Y = RO - RA*COS(THETA) + YO
C      ELSEIF (N.EQ.1) THEN
C        XN = X - XO
C        YN = RO - Y + YO
C        RA = SQRT(XN*XN+YN*YN)
C        ALAT = 0.5*(SC-(RA*SN/R)**2)/SN
C        ALAT = ASIN(ALAT)
C        IF (ABS(XN).LE.0.0) THEN
C          THETA = 0.
C        ELSE
C          IF (ABS(YN).LE.0.0) THEN
C            THETA = PI*0.5
C            IF (XN.LT.0.0) THETA = -THETA
C          ELSE
C            THETA = ATAN2(XN,YN)
C          ENDIF
C        ENDIF
C        ALON = THETA/SN + OLON
C        ALAT = ALAT*RADDEG
C        ALON = ALON*RADDEG
C      ENDIF
C      RETURN
C
C... ERROR.

```

```

C
  999 CONTINUE
    PRINT *, 'OVER [ ALBPROJ ]'
    PRINT *, '(SLAT1,SLAT2) : ', SLAT1*RADDEG,SLAT2*RADDEG
    PRINT *, '(OLAT ,OLON ) : ', OLAT*RADDEG,OLON*RADDEG
    PRINT *, '(ALAT ,ALON ) : ', ALAT,ALON
    IERR = 1
    RETURN
    END

C*****
  SUBROUTINE ALBEINIT
C*****
C  Albers Equal-area Conic projection Initialize
C
C-----
  COMMON /ALBERC/ R,SLAT1,SLAT2,SN,SC,RO,OLAT,OLON,XO,YO,IFIRST
  COMMON /CALCUL/ PI,DEGRAD,RADDEG

  PI = ASIN(1.)*2.
  DEGRAD = PI/180.
  RADDEG = 180./PI

  R = 6370.1846      ! [km]      : Earth Radius
  SLAT1 = 30.0       ! [degree] : Standard Latitude 1
  SLAT2 = 60.0       ! [degree] : Standard Latitude 2
  SLON = 125.0       ! [degree] : Standard Longitude
  OLAT = 35.0        ! [degree] : Latitude of known point in map
  OLON = 125.0       ! [degree] : Longitude of known point in map
  XO = 0.0           ! [grid]  : X-coordinate of known point
  YO = 0.0           ! [grid]  : Y-coordinate of known point
  DD = 4.0           ! [km]    : Grid distance in map

  SLAT1 = SLAT1*DEGRAD
  SLAT2 = SLAT2*DEGRAD
  SLON = SLON*DEGRAD
  OLAT = OLAT*DEGRAD
  OLON = OLON*DEGRAD
  R = R/DD

  RETURN
  END

```

카. Stereographic projection

```

C*****
C      SUBROUTINE STERPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Stereographic projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y      : (x,y) coordinate in map [grid]
C      * N = 0   : (lat,lon) --> (x,y)
C      * N = 1   : (x,y) --> (lat,lon)
C-----
C      COMMON /STEREO/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C        CALL STERINIT
C        AK = 1./SIN(SLAT)*SIN(OLAT)+COS(SLAT)*COS(OLAT)*COS(OLON-SLON)
C        IF (AK.LE.0.0) THEN
C          PRINT *, 'ERROR [ STERPROJ ]'
C          PRINT *, '(SLAT,SLON) : ',SLAT*RADDEG,SLON*RADDEG
C          PRINT *, '(OLAT,OLON) : ',OLAT*RADDEG,OLON*RADDEG
C          STOP
C        ENDIF
C        AK = 2./AK
C        XO = R*AK*COS(OLAT)*SIN(OLON-SLON) - XO
C        YO = R*AK*(COS(SLAT)*SIN(OLAT)
C          * -SIN(SLAT)*COS(OLAT)*COS(OLON-SLON)) - YO
C        IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C        BLAT = ALAT*DEGRAD
C        BLON = ALON*DEGRAD - SLON
C        AK = 1./SIN(SLAT)*SIN(BLAT)+COS(SLAT)*COS(BLAT)*COS(BLON)
C        IF (AK.LE.0.0) GOTO 999
C        AK = 2./AK
C        X = R*AK*COS(BLAT)*SIN(BLON) - XO
C        Y = R*AK*(COS(SLAT)*SIN(BLAT)
C          * -SIN(SLAT)*COS(BLAT)*COS(BLON)) - YO
C      ELSEIF (N.EQ.1) THEN
C        XN = X + XO
C        YN = Y + YO
C        RA = XN*XN + YN*YN
C        IF (RA.LE.0.0) THEN
C          AC = 0.
C          ALAT = SLAT
C        ELSE
C          RA = SQRT(RA)
C          AC = 2.*ATAN(0.5*RA/R)
C          ALAT = COS(AC)*SIN(SLAT) + YN*SIN(AC)*COS(SLAT)/RA
C          ALAT = ASIN(ALAT)
C        ENDIF
C        A1 = XN*SIN(AC)
C        A2 = RA*COS(SLAT)*COS(AC)-YN*SIN(SLAT)*SIN(AC)
C        IF (ABS(A1).LE.0.0) THEN
C          THETA = 0.
C        ELSEIF (ABS(A2).LE.0.0) THEN
C          THETA = PI*0.5
C          IF (A2.LT.0.0) THETA = -THETA
C        ELSE
C          THETA = ATAN2(A1,A2)
C        ENDIF
C        ALON = THETA + SLON
C        ALAT = ALAT*RADDEG

```

```

        ALON = ALON*RADDEG
    ENDIF
    RETURN
C
C... ERROR.
C
999 CONTINUE
    PRINT *, 'OVER [ STERPROJ ]'
    PRINT *, '(SLAT,SLON) : ', SLAT*RADDEG, SLON*RADDEG
    PRINT *, '(OLAT,OLON) : ', OLAT*RADDEG, OLON*RADDEG
    PRINT *, '(ALAT,ALON) : ', ALAT, ALON
    IERR = 1
    RETURN
    END

C*****
SUBROUTINE STERINIT
C*****
C Stereographic projection Initialize
C
C-----
COMMON /STEREO/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
COMMON /CALCUL/ PI,DEGRAD,RADDEG

PI = ASIN(1.)*2.
DEGRAD = PI/180.
RADDEG = 180./PI

R = 6370.1846      ! [km]      : Earth Radius
SLAT = 90.0        ! [degree] : Standard Latitude
SLON = 125.0        ! [degree] : Standard Longitude
OLAT = 35.0         ! [degree] : Latitude of known point in map
OLON = 125.0        ! [degree] : Longitude of known point in map
XO = 0.0            ! [grid]   : X-coordinate of known point
YO = 0.0            ! [grid]   : Y-coordinate of known point
DD = 4.0            ! [km]     : Grid distance in map

SLAT = SLAT*DEGRAD
SLON = SLON*DEGRAD
OLAT = OLAT*DEGRAD
OLON = OLON*DEGRAD
R = R/DD

C
C if SLAT=90 degree and 60N (IPOLAR=1)
C
IPOLAR = 1
IF (IPOLAR.EQ.1) THEN
    R = R*(1.0+SIN(60.0*DEGRAD))*0.5
ENDIF

RETURN
END

```

4. Orthographic projection

```

C*****
C      SUBROUTINE ORTHPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Orthographic projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y      : (x,y) coordinate in map [grid]
C      * N = 0   : (lat,lon) --> (x,y)
C      * N = 1   : (x,y) --> (lat,lon)
C-----
C      COMMON /ORTHOG/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C          CALL ORTHINIT
C          XO = R*COS(OLAT)*SIN(OLON-SLON) - XO
C          CH = SIN(SLAT)*SIN(OLAT)+COS(SLAT)*COS(OLAT)*COS(OLON-SLON)
C          IF (CH.LT.-0.00001) THEN
C              PRINT *, 'ERROR [ ORTHPROJ ]'
C              PRINT *, '(SLAT,SLON) : ',SLAT*RADDEG,SLON*RADDEG
C              PRINT *, '(OLAT,OLON) : ',OLAT*RADDEG,OLON*RADDEG
C              STOP
C          ELSE
C              YO = R*(COS(SLAT)*SIN(OLAT)
C              *      -SIN(SLAT)*COS(OLAT)*COS(OLON-SLON)) - YO
C          ENDIF
C          IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C          BLAT = ALAT*DEGRAD
C          BLON = ALON*DEGRAD - SLON
C          CH = SIN(SLAT)*SIN(BLAT)+COS(SLAT)*COS(BLAT)*COS(BLON)
C          IF (CH.LT.-0.00001) GOTO 999
C          X = R*COS(BLAT)*SIN(BLON) - XO
C          Y = R*(COS(SLAT)*SIN(BLAT)
C          *      -SIN(SLAT)*COS(BLAT)*COS(BLON)) - YO
C      ELSEIF (N.EQ.1) THEN
C          XN = X + XO
C          YN = Y + YO
C          RA = XN*XN + YN*YN
C          IF (RA.LE.0.0) THEN
C              AC = 0.
C              ALAT = SLAT
C          ELSE
C              RA = SQRT(RA)
C              AC = ASIN(RA/R)
C              ALAT = COS(AC)*SIN(SLAT) + YN*SIN(AC)*COS(SLAT)/RA
C              ALAT = ASIN(ALAT)
C          ENDIF
C          A1 = XN*SIN(AC)
C          A2 = RA*COS(SLAT)*COS(AC)-YN*SIN(SLAT)*SIN(AC)
C          IF (ABS(A1).LE.0.0) THEN
C              THETA = 0.
C          ELSEIF (ABS(A2).LE.0.0) THEN
C              THETA = PI*0.5
C              IF (A2.LT.0.0) THETA = -THETA
C          ELSE
C              THETA = ATAN2(A1,A2)
C          ENDIF
C          ALON = THETA + SLON
C          ALAT = ALAT*RADDEG
C          ALON = ALON*RADDEG

```



```

        ENDIF
        RETURN
C
C... ERROR.
C
      999 CONTINUE
        PRINT *, 'OVER [ ORTHPROJ ]'
        PRINT *, '(SLAT,SLON) : ', SLAT*RADDEG, SLON*RADDEG
        PRINT *, '(OLAT,OLON) : ', OLAT*RADDEG, OLON*RADDEG
        PRINT *, '(ALAT,ALON) : ', ALAT, ALON
        IERR = 1
        RETURN
        END

C*****
      SUBROUTINE ORTHINIT
C*****
C      Orthographic projection Initialize
C
C-----
      COMMON /ORTHOG/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
      COMMON /CALCUL/ PI,DEGRAD,RADDEG

      PI = ASIN(1.)*2.
      DEGRAD = PI/180.
      RADDEG = 180./PI

      R = 6370.1846      ! [km]      : Earth Radius
      SLAT = 90.0        ! [degree] : Standard Latitude
      SLON = 125.0       ! [degree] : Standard Longitude
      OLAT = 35.0        ! [degree] : Latitude of known point in map
      OLON = 125.0       ! [degree] : Longitude of known point in map
      XO = 0.0           ! [grid]  : X-coordinate of known point
      YO = 0.0           ! [grid]  : Y-coordinate of known point
      DD = 4.0           ! [km]    : Grid distance in map

      SLAT = SLAT*DEGRAD
      SLON = SLON*DEGRAD
      OLAT = OLAT*DEGRAD
      OLON = OLON*DEGRAD
      R = R/DD

      RETURN
      END

```

Ⅱ. Azimuthal equidistant projection

```

C*****
C      SUBROUTINE AZEDPROJ(ALAT,ALON,X,Y,N,IERR)
C*****
C      *** Azimuthal Equidistant projection ***
C
C      ALAT, ALON : (latitude,longitude) at earth [degree]
C      X, Y      : (x,y) coordinate in map [grid]
C      * N = 0   : (lat,lon) --> (x,y)
C      * N = 1   : (x,y) --> (lat,lon)
C-----
C      COMMON /AZIEQD/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
C      COMMON /CALCUL/ PI,DEGRAD,RADDEG
C
C... INITIALIZE.
C
C      IF (IFIRST.EQ.0) THEN
C          CALL AZEDINIT
C          AK = SIN(SLAT)*SIN(OLAT)+COS(SLAT)*COS(OLAT)*COS(OLON-SLON)
C          IF (AK.LT.-0.99999) GOTO 999
C          IF (AK.GT.0.99999) THEN
C              AK = 1.
C          ELSE
C              AK = ACOS(AK)
C              AK = AK/SIN(AK)
C          ENDIF
C          XO = R*AK*COS(OLAT)*SIN(OLON-SLON) - XO
C          YO = R*AK*(COS(SLAT)*SIN(OLAT)
C      *      -SIN(SLAT)*COS(OLAT)*COS(OLON-SLON)) - YO
C          IFIRST = 1
C      ENDIF
C
C... CONVERT.
C
C      IF (N.EQ.0) THEN
C          BLAT = ALAT*DEGRAD
C          BLON = ALON*DEGRAD - SLON;
C          AK = SIN(SLAT)*SIN(BLAT)+COS(SLAT)*COS(BLAT)*COS(BLON)
C          IF (AK.LT.-0.99999) GOTO 999
C          IF (AK.GT.0.99999) THEN
C              AK = 1.
C          ELSE
C              AK = ACOS(AK)
C              AK = AK/SIN(AK)
C          ENDIF
C          X = R*AK*COS(BLAT)*SIN(BLON) - XO
C          Y = R*AK*(COS(SLAT)*SIN(BLAT)
C      *      -SIN(SLAT)*COS(BLAT)*COS(BLON)) - YO
C      ELSEIF (N.EQ.1) THEN
C          XN = X + XO
C          YN = Y + YO
C          RA = XN*XN + YN*YN
C          IF (RA.LE.0.0) THEN
C              AC = 0.
C              ALAT = SLAT
C          ELSE
C              RA = SQRT(RA)
C              AC = RA/R
C              ALAT = COS(AC)*SIN(SLAT) + YN*SIN(AC)*COS(SLAT)/RA
C              ALAT = ASIN(ALAT)
C          ENDIF
C          A1 = XN*SIN(AC)
C          A2 = RA*COS(SLAT)*COS(AC)-YN*SIN(SLAT)*SIN(AC)
C          IF (ABS(A1).LE.0.0) THEN
C              THETA = 0.
C          ELSEIF (ABS(A2).LE.0.0) THEN
C              THETA = PI*0.5
C          IF (A2.LT.0.0) THETA = -THETA

```

```

        ELSE
            THETA = ATAN2(A1,A2)
        ENDIF
        ALON = THETA + SLON
        ALAT = ALAT*RADDEG
        ALON = ALON*RADDEG
    ENDIF
    RETURN
C
C... ERROR.
C
    999 CONTINUE
    PRINT *, 'OVER [ AZEDPROJ ]'
    PRINT *, '(SLAT,SLON) :', SLAT*RADDEG, SLON*RADDEG
    PRINT *, '(OLAT,OLON) :', OLAT*RADDEG, OLON*RADDEG
    PRINT *, '(ALAT,ALON) :', ALAT, ALON
    IERR = 1
    RETURN
    END

C*****
SUBROUTINE AZEDINIT
C*****
C Azimuthal Equidistant projection Initialize
C
C-----
COMMON /AZIEQD/ R,SLAT,SLON,OLAT,OLON,XO,YO,IFIRST
COMMON /CALCUL/ PI,DEGRAD,RADDEG

PI = ASIN(1.)*2.
DEGRAD = PI/180.
RADDEG = 180./PI

R = 6370.1846      ! [km]      : Earth Radius
SLAT = 90.0        ! [degree] : Standard Latitude
SLON = 125.0        ! [degree] : Standard Longitude
OLAT = 35.0         ! [degree] : Latitude of known point in map
OLON = 125.0        ! [degree] : Longitude of known point in map
XO = 0.0            ! [grid]   : X-coordinate of known point
YO = 0.0            ! [grid]   : Y-coordinate of known point
DD = 4.0            ! [km]     : Grid distance in map

SLAT = SLAT*DEGRAD
SLON = SLON*DEGRAD
OLAT = OLAT*DEGRAD
OLON = OLON*DEGRAD
R = R/DD

RETURN
END

```

2. C 버전

가. Mercator Projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"

/*****
 *
 * [ Mercator Projection ]
 *
 * o lon, lat : (longitude,latitude) at earth [degree]
 * o x, y      : (x,y) coordinate in map [grid]
 * o code = 0 : (lon,lat) --> (x,y)
 *             1 : (x,y) --> (lon,lat)
 *
 *****/
int mercproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct merc_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slon, slat, olon, olat, xo, yo;
    double        alon, alat, xn;

    if (code == 0 && fabs(*lat) > 89.9) return -1;
    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        slon = map.slon * DEGRAD;
        slat = map.slat * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        re = (map.Re/map.grid)*cos(slat);
        xn = olon - slon;
        if (xn > PI) xn -= 2.0*PI;
        else if (xn < -PI) xn += 2.0*PI;
        xo = re*xn - map.xo;
        yo = re*log( tan( PI*0.25 + olat*0.5 ) ) - map.yo;
        map.first = 1;
    }

    if (code == 0) {
        xn = (*lon)*DEGRAD - slon;
        if (xn > PI) xn -= 2.0*PI;
        else if (xn < -PI) xn += 2.0*PI;
        (*x) = re*xn - xo;
        (*y) = re*log( tan( PI*0.25 + (*lat)*DEGRAD*0.5 ) ) - yo;
    } else {
        alat = PI*0.5 - 2.0*atan(exp(-((*y)+yo)/re));
        alon = ((*x)+xo)/re + slon;
        *lat = alat * RADDEG;
        *lon = alon * RADDEG;
    }
    return 0;
}
```

4. Transveres mercator projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"

/*****
 *
 * [ Transveres Mercator Projection ]
 *
 * o lon, lat : (longitude,latitude) at earth [degree]
 * o x, y      : (x,y) coordinate in map [grid]
 * o code = 0 : (lon,lat) --> (x,y)
 * o          1 : (x,y) --> (lon,lat)
 *
 *****/
int trnmproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct trnm_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slon, slat, olon, olat, xo, yo;
    double        alon, alat, xn, yn, b, theta;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slon = map.slون * DEGRAD;
        slat = map.slat * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        b = cos(olat) * sin(olon-slon);
        if (fabs(b) >= 0.99999) return -1;
        xo = 0.5*re*log( (1.0+b)/(1.0-b) ) - map.xo;
        if (fabs(olat) >= PI*0.5) {
            yn = olat;
            if (fabs(olon-slon) > PI*0.5) yn = -yn;
        } else if (fabs(cos(olon-slon)) <= 0.0) {
            yn = PI*0.5;
            if (olat > 0.0) yn = -yn;
            if (olat == 0.0) yn = 0.0;
        } else {
            yn = atan2( tan(olat), cos(olon-slon) );
        }
        yo = re*(yn - slat) - map.yo;
        map.first = 1;
    }

    if (code == 0) {
        alat = (*lat)*DEGRAD;
        alon = (*lon)*DEGRAD - slon;
        b = cos(alat) * sin(alon);
        if (fabs(b) >= 0.99999) return -1;
        *x = 0.5*re*log((1.0+b)/(1.0-b)) - xo;
        if (fabs(*lat) >= 90.0) {
            yn = PI*0.5;
            if (alat < 0.0) yn = -yn;
        } else if (fabs(cos(alon)) <= 0.0) {
            yn = PI*0.5;
            if ((*lat) < 0.0) yn = -yn;
            if ((*lat) == 0.0) yn = 0.0;
        } else {

```

```

        yn = atan2( tan(alat), cos(alon) );
    }
    *y = re*(yn - slat) - yo;
} else {
    xn = (*x) + xo;
    yn = (*y) + yo;
    b = yn/re + slat;
    alat = sin(b) / cosh(xn/re);
    *lat = asin(alat) * RADDEG;
    if( fabs(cos(b)) <= 0.00001 ) {
        theta = PI*0.5;
        if( xn < 0.0 ) theta = -theta;
        if( xn == 0.0 ) theta = 0.0;
    } else {
        theta = atan2( sinh(xn/re), cos(b) );
    }
    *lon = (theta + slon) * RADDEG;
}
return 0;
}

```

다. Oblique mercator projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"

/*****
*
* [ Oblique Mercator Projection ]
*
* o lon, lat : (longitude,latitude) at earth [degree]
* o x, y      : (x,y) coordinate in map [grid]
* o code = 0 : (lon,lat) --> (x,y)
*             1 : (x,y) --> (lon,lat)
*
*****/
int oblmproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct oblmp_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slat, slon, olon, olat, xo, yo;
    double xn, yn, alat, alon, a, b, c, b1;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slon = map.slon * DEGRAD;
        slat = map.slat * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        b = sin(olon - slon);
        if (fabs(olat) >= 0.5*PI) {
            xn = 0.5*PI;
            if (olat*b < 0.0) xn = -xn;
        } else {
            c = tan(olat)*cos(slat) - sin(slat)*cos(olon - slon);
            if (fabs(c) <= 0.0) {
                xn = 0.0;
            } else {
                if (fabs(b) <= 0.00001) {
                    xn = 0.5*PI;
                    if (c*b < 0.0) xn = -xn;
                } else {
                    xn = atan2(c, b);
                }
            }
        }
        xo = re*xn - map.xo;
        a = sin(slat)*sin(olat) + cos(slat)*cos(olat)*cos(olon-slon);
        if (fabs(a) >= 0.99999) return -1;
        yo = 0.5*re*log((1.0+a)/(1.0-a)) - map.yo;
        map.first = 1;
    }

    if (code == 0) {
        alat = (*lat)*DEGRAD;
        alon = (*lon)*DEGRAD - slon;
        b = sin(alon);
        if (fabs(*lat) >= 90.0) {
            xn = 0.5*PI;
            if (*lat < 0.0) xn = -xn;
        } else {

```

```

c = tan(alat)*cos(slat) - sin(slat)*cos(alon);
if (fabs(c) <= 0.0) {
    xn = 0.0;
} else {
    if (fabs(b) <= 0.00001) {
        b1 = cos(alon);
        if ((b1 > 0.0 && alat < slat) || (b1 < 0.0 && alat < -slat)) {
            xn = -0.5*PI;
        } else {
            xn = 0.5*PI;
        }
    } else {
        xn = atan2(c, b);
    }
}
}
*x = re*xn - xo;
a = sin(slat)*sin(alat) + cos(slat)*cos(alat)*cos(alon);
if (fabs(a) >= 0.99999) return -1;
*y = 0.5*re*log((1.0+a)/(1.0-a)) - yo;
} else {
    xn = (*x + xo)/re;
    yn = (*y + yo)/re;
    *lat = sin(slat)*tanh(yn) + cos(slat)*sin(xn)/cosh(yn);
    *lat = asin(*lat) * RADDEG;
    c = sin(slat)*sin(xn) - cos(slat)*sinh(yn);
    b = cos(xn);
    if (fabs(c) <= 0.0) {
        *lon = 0.0;
    } else if (fabs(b) <= 0.00001) {
        *lon = 0.5*PI;
        if (c*b < 0.0) *lon = -(*lon);
    } else {
        *lon = atan2(c, b) + 0.5*PI;
    }
    *lon = (*lon + slon) * RADDEG;
}
return 0;
}

```


라. Miller cylindrical projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"

/*****
*
* [ Miller Cylindrical Projection ]
*
* o lon, lat : (longitude,latitude) at earth [degree]
* o x, y      : (x,y) coordinate in map [grid]
* o code = 0 : (lon,lat) --> (x,y)
* o         1 : (x,y) --> (lon,lat)
*
*****/
int milmproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct milm_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slon, olon, olat, xo, yo;
    double xn;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slon = map.slon * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        xn = olon - slon;
        if (xn > PI) xn -= 2.0*PI;
        if (xn < -PI) xn += 2.0*PI;
        xo = re*xn - map.xo;
        yo = re*log(tan(0.25*PI + 0.4*olat))/0.8 - map.yo;
        map.first = 1;
    }

    if (code == 0) {
        xn = (*lon)*DEGRAD - slon;
        if (xn > PI) xn -= 2.0*PI;
        if (xn < -PI) xn += 2.0*PI;
        *x = re*xn - xo;
        *y = re*log(tan(0.25*PI + (*lat)*DEGRAD*0.4))/0.8 - yo;
    } else {
        *lat = (2.5*atan(exp(0.8*(*y + yo)/re)) - 5.0*PI/8.0) * RADDEG;
        *lon = ((*x + xo)/re + slon) * RADDEG;
    }
    return 0;
}
```

04. Equidistant cylindrical projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"

/*****
 *
 * [ Equidistant Cylindrical Projection ]
 *
 *   o lon, lat : (longitude,latitude) at earth [degree]
 *   o x, y     : (x,y) coordinate in map [grid]
 *   o code = 0 : (lon,lat) --> (x,y)
 *             1 : (x,y) --> (lon,lat)
 *
 *****/
int eqdcproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct eqdc_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slat, slon, olon, olat, xo, yo;
    double xn;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slon = map.slون * DEGRAD;
        slat = map.slat * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        xn = olon - slon;
        if (xn > PI) xn -= 2.0*PI;
        if (xn < -PI) xn += 2.0*PI;
        xo = re*xn*cos(slat) - map.xo;
        yo = map.yo;
        map.first = 1;
    }

    if (code == 0) {
        xn = (*lon)*DEGRAD - slon;
        if (xn > PI) xn -= 2.0*PI;
        if (xn < -PI) xn += 2.0*PI;
        *x = re*xn*cos(slat) - xo;
        *y = re*((*lat)*DEGRAD - olat) + yo;
    } else {
        *lat = ((*y - yo)/re + olat) * RADDEG;
        *lon = ((*x + xo)/(re*cos(slat)) + slon) * RADDEG;
    }
    return 0;
}
```

바. Sinusoidal projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"

/*****
*
* [ Sinusoidal Projection ]
*
* o lon, lat : (longitude,latitude) at earth [degree]
* o x, y      : (x,y) coordinate in map [grid]
* o code = 0 : (lon,lat) --> (x,y)
* o         1 : (x,y) --> (lon,lat)
*
*****/
int sinuproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct sinu_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slon, olon, olat, xo, yo;
    double        alat, alon;

    if( map.first == 0 ) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slon = map.slون * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        xo = re*(olon-slon)*cos(olat) - map.xo;
        yo = map.yo;
        map.first = 1;
    }

    if (code == 0) {
        alat = (*lat)*DEGRAD;
        alon = (*lon)*DEGRAD - slon;
        if (alon < -PI) alon += 2.0*PI;
        if (alon > PI) alon -= 2.0*PI;
        *x = re*alon*cos(alat) - xo;
        *y = re*(alat - olat) + yo;
    } else {
        *lat = (*y - yo)/re + olat;
        *lon = ((*x + xo)/(re*cos(*lat)) + slon) * RADDEG;
        *lat = (*lat) * RADDEG;
    }
    return 0;
}
```

사. Van der grinten projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"

/*****
*
* [ Van der grinten Projection ]
*
* o lon, lat : (longitude,latitude) at earth [degree]
* o x, y      : (x,y) coordinate in map [grid]
* o code = 0 : (lon,lat) --> (x,y)
*             1 : (x,y) --> (lon,lat)
*
*****/
int vandproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct vand_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slat, slon, olon, olat, xo, yo;
    double xn, yn, alat, alon, theta, a, g, p;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slon = map.slون * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        if (olat == 0.0) {
            xn = re*fabs(olon - slon);
            yn = 0.0;
        } else if (olon == slon) {
            theta = asin(fabs(2.0*olat/PI));
            xn = 0.0;
            yn = re*tan(theta*0.5)*PI;
        } else {
            theta = asin(fabs(2.0*olat/PI));
            a = 0.5*fabs(PI/(olon-slon) - (olon-slon)/PI);
            g = cos(theta)/(sin(theta) + cos(theta) - 1.0);
            p = g*(2.0/cos(theta) - 1.0);
            xn = a*(g-p*p) + sqrt(a*a*pow((g-p*p), 2) - (p*p+a*a)*(g*g-p*p));
            xn = PI*re*xn/(a*a+p*p);
            yn = PI*re*sqrt(1.0 - pow((xn/PI*re), 2) - 2.0*a*fabs(xn/(PI*re)));
        }
        if (olon < slon) xn = -xn;
        if (olat < 0.0) yn = -yn;
        xo = xn - map.xo;
        yo = yn - map.yo;
        map.first = 1;
    }

    if (code == 0) {
        alat = (*lat)*DEGRAD;
        alon = (*lon)*DEGRAD - slon;
        if (alon < -PI) alon += 2.0*PI;
        if (alon > PI) alon -= 2.0*PI;
        if (alat == 0.0) {
            *x = re*fabs(alon);
            *y = 0.0;
        } else if (fabs(alon) == 0.0) {
            *x = 0.0;

```

```

        theta = asin(fabs(2.0*alat/PI));
        *y = PI*re*tan(theta*0.5);
    } else {
        theta = asin(fabs(2.0*alat/PI));
        a = 0.5*fabs(PI/alon - alon/PI);
        g = cos(theta) / (sin(theta) + cos(theta) - 1.0);
        p = g*(2.0/sin(theta) - 1.0);
        *x = a*(g-p*p) + sqrt(a*a*pow((g-p*p), 2) - (p*p+a*a)*(g*g-p*p));
        *x = PI*re*(*x)/(a*a+p*p);
        *y = 1.0 - pow(*x/(PI*re),2) - 2.0*a*fabs(*x/(PI*re));
        if (*y <= 0.0) {
            *y = 0.0;
        } else {
            *y = PI*re*sqrt(*y);
        }
    }
    if (alon < 0.0) *x = -(*x);
    if (alat < 0.0) *y = -(*y);
    *x -= xo;
    *y -= yo;
} else {
    xn = *x + xo;
    yn = *y + yo;
    if (xn == 0.0) {
        theta = 2.0*atan(yn/(PI*re));
        *lat = 0.5*PI*sin(theta);
        *lon = olon;
    } else if (yn == 0.0) {
        *lat = 0.0;
        *lon = xn/re + olon;
    } else {
        return -1;
    }
    *lat *= RADDEG;
    *lon *= RADDEG;
}
return 0;
}

```

04. Lambert conformal conic projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"
/*****
*
* [ Lambert Conformal Conic Projection ]
*
* o lon, lat : (longitude,latitude) at earth [degree]
* o x, y      : (x,y) coordinate in map [grid]
* o code = 0 : (lon,lat) --> (x,y)
*             1 : (x,y) --> (lon,lat)
*
*****/
int lamcproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct lamc_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, olon, olat, sn, sf, ro;
    double        slat1, slat2, alon, alat, xn, yn, ra, theta;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slat1 = map.slat1 * DEGRAD;
        slat2 = map.slat2 * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        sn = tan(PI*0.25 + slat2*0.5)/tan(PI*0.25 + slat1*0.5);
        sn = log(cos(slat1)/cos(slat2))/log(sn);
        sf = tan(PI*0.25 + slat1*0.5);
        sf = pow(sf,sn)*cos(slat1)/sn;
        ro = tan(PI*0.25 + olat*0.5);
        ro = re*sf/pow(ro,sn);
        map.first = 1;
    }

    if (code == 0) {
        ra = tan(PI*0.25+(*lat)*DEGRAD*0.5);
        ra = re*sf/pow(ra,sn);
        theta = (*lon)*DEGRAD - olon;
        if (theta > PI) theta -= 2.0*PI;
        if (theta < -PI) theta += 2.0*PI;
        theta *= sn;
        *x = (float)(ra*sin(theta)) + map.xo;
        *y = (float)(ro - ra*cos(theta)) + map.yo;
    } else {
        xn = *x - map.xo;
        yn = ro - *y + map.yo;
        ra = sqrt(xn*xn+yn*yn);
        if (sn < 0.0) -ra;
        alat = pow((re*sf/ra),(1.0/sn));
        alat = 2.0*atan(alat) - PI*0.5;
        if (fabs(xn) <= 0.0) {
            theta = 0.0;
        } else {
            if (fabs(yn) <= 0.0) {
                theta = PI*0.5;
                if ( xn < 0.0 ) -theta;
            } else
                theta = atan2(xn,yn);
        }
        alon = theta/sn + olon;
        *lat = (float)(alat*RADDEG);
        *lon = (float)(alon*RADDEG);
    }
    return 0;
}
```

자. Lambert azimuthal equal-area projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"
/*****
*
* [ Lambert Azimuthal Equal-Area Projection ]
*
* o lon, lat : (longitude,latitude) at earth [degree]
* o x, y      : (x,y) coordinate in map [grid]
* o code = 0 : (lon,lat) --> (x,y)
*             1 : (x,y) --> (lon,lat)
*
*****/
int lameproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct lame_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slon, slat, olon, olat, xo, yo;
    double ak, alon, alat, xn, yn, ra, ac, a1, a2, theta;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slon = map.slون * DEGRAD;
        slat = map.slat * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        ak = 1.0 + sin(slat)*sin(olat) + cos(slat)*cos(olat)*cos(olon-slon);
        if (ak <= 0.00001) {
            xo = -map.xo;
            yo = -map.yo;
        } else {
            ak = sqrt(2.0/ak);
            xo = re*ak*cos(olat)*sin(olon-slon) - map.xo;
            yo = re*ak*(cos(slat)*sin(olat) - sin(slat)*cos(olat)*cos(olon-slon)) - map.yo;
        }
        map.first = 1;
    }

    if (code == 0) {
        alat = (*lat)*DEGRAD;
        alon = (*lon)*DEGRAD - slon;
        ak = 1.0 + sin(slat)*sin(alat) + cos(slat)*cos(alat)*cos(alon);
        if (ak <= 0.00001) return -1;
        ak = sqrt(2.0/ak);
        *x = re*ak*cos(alat)*sin(alon) - xo;
        *y = re*ak*(cos(slat)*sin(alat) - sin(slat)*cos(alat)*cos(alon)) - yo;
    } else {
        xn = *x + xo;
        yn = *y + yo;
        ra = xn*xn + yn*yn;
        if (ra <= 0.0) {
            ac = 0.0;
            *lat = slat * RADDEG;
        } else {
            ra = sqrt(ra);
            ac = 2.0*asin(0.5*ra/re);
            *lat = cos(ac)*sin(slat) + yn*sin(ac)*cos(slat)/ra;
            *lon = asin(*lat) * RADDEG;
        }
    }
}
```

```

a1 = xn*sin(ac);
a2 = ra*cos(slat)*cos(ac) - yn*sin(slat)*sin(ac);
if (fabs(a1) <= 0.0) {
    theta = 0.0;
} else if (fabs(a2) <= 0.0) {
    theta = 0.5*PI;
    if (a2 < 0.0) theta = -theta;
} else {
    theta = atan2(a1, a2);
}
*lon = (theta + slon)*RADDEG;
}
return 0;
}

```


차. albers equal-area conic projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"

/*****
*
* [ Albers Equal-area Conic projection ]
*
* o lon, lat : (longitude,latitude) at earth [degree]
* o x, y      : (x,y) coordinate in map [grid]
* o code = 0 : (lon,lat) --> (x,y)
*           1 : (x,y) --> (lon,lat)
* o map      : map parameter
*
*****/
int albeproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct albe_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slat1, slat2, sn, sc, ro, olon, olat;
    double alon, alat, xn, yn, ra, theta;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slat1 = map.slat1 * DEGRAD;
        slat2 = map.slat2 * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        sn = (sin(slat1) + sin(slat2)) * 0.5;
        if (fabs(sn) <= 0.0) return -1;
        sc = 1.0 + sin(slat1) * sin(slat2);
        if (sn*olat < 0.0 && fabs(alat) > 89.9*DEGRAD) return -1;
        ro = re * sqrt(sc - 2.0*sn*sin(olat))/sn;
        map.first = 1;
    }

    if (code == 0) {
        if ((*lat)*sn < 0.0 && fabs(*lat) > 89.9) return -1;
        ra = re * sqrt(sc - 2.0*sn*sin((*lat)*DEGRAD))/sn;
        theta = (*lon)*DEGRAD - olon;
        if (theta > PI) theta -= 2.0*PI;
        if (theta < -PI) theta += 2.0*PI;
        theta *= sn;
        *x = ra * sin(theta) + map.xo;
        *y = ro - ra * cos(theta) + map.yo;
    } else {
        xn = *x - map.xo;
        yn = ro - *y + map.yo;
        ra = sqrt(xn*xn + yn*yn);
        *lat = 0.5*(sc - pow(ra*sn/re, 2))/sn;
        *lat = asin(*lat) * RADDEG;
        if (fabs(xn) <= 0.0) {
            theta = 0.0;
        } else {
            if (fabs(yn) <= 0.0) {
                theta = 0.5 * PI;
                if (xn < 0.0) theta = -theta;
            } else {
                theta = atan2(xn, yn);
            }
        }
        *lon = (theta/sn + olon) * RADDEG;
    }
    return 0;
}
```

카. Stereographic projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"

/*****
*
* [ Stereographic Projection ]
*
* o lon, lat : (longitude,latitude) at earth [degree]
* o x, y      : (x,y) coordinate in map [grid]
* o code = 0 : (lon,lat) --> (x,y)
*             1 : (x,y) --> (lon,lat)
* o map      : map parameter
*
*****/
int sterproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct ster_parameter map;
{
    static int first = 0, polar = 1;
    static double re, slat, slon, olon, olat, xo, yo;
    static double PI, DEGRAD, RADDEG;
    double Re, alon, alat, ak, dd, xn, yn, a1, a2, ra, ac, theta;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slon = map.slon * DEGRAD;
        slat = map.slat * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        /* if slat = 90N and 60N 꺾임 */
        if( map.polar == 1 ) {
            re = re * ( 1.0 + sin( 60.0*DEGRAD ) ) * 0.5;
        }
        ak = 1.0 + sin(slat)*sin(olat) + cos(slat)*cos(olat)*cos(olon-slon);
        if( ak <= 0.0 ) return -1;
        ak = 2.0/ak;
        xo = re*ak*cos(olat)*sin(olon-slon) - map.xo;
        yo = re*ak*( cos(slat)*sin(olat) - sin(slat)*cos(olat)*cos(olon-slon) ) - map.yo;
        map.first = 1;
    }

    if (code == 0) {
        alat = (*lat)*DEGRAD;
        alon = (*lon)*DEGRAD - slon;
        ak = 1.0 + sin(slat)*sin(alat) + cos(slat)*cos(alat)*cos(alon);
        if( ak <= 0.0 ) {
            return -1;
        } else {
            ak = 2.0/ak;
            (*x) = re*ak*cos(alat)*sin(alon) - xo;
            (*y) = re*ak*( cos(slat)*sin(alat) - sin(slat)*cos(alat)*cos(alon) ) - yo;
        }
    } else {
        xn = (*x) + xo;
        yn = (*y) + yo;
        ra = xn*xn + yn*yn;
        if( ra <= 0.0 ) {
            ac = 0.0;
            alat = slat;
        }
    }
}
```

```

    } else {
        ra = sqrt(ra);
        ac = 2.0*atan(0.5*ra/re);
        alat = cos(ac)*sin(slat) + yn*sin(ac)*cos(slat)/ra;
        alat = asin(alat);
    }
    a1 = xn*sin(ac);
    a2 = ra*cos(slat)*cos(ac) - yn*sin(slat)*sin(ac);
    if (fabs(a1) <= 0.0) {
        theta = 0.0;
    } else if( abs(a2) <= 0.0 ) {
        theta = PI*0.5;
        if( a2 < 0.0 ) theta = -theta;
    } else {
        theta = atan2(a1,a2);
    }
    alon = theta + slon;
    *lon = alon*RADDEG;
    *lat = alat*RADDEG;
}
return 0;
}

```

E4. Orthographic projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"
/*****
*
* [ Orthographic Projection ]
*
* o lon, lat : (longitude,latitude) at earth [degree]
* o x, y      : (x,y) coordinate in map [grid]
* o code = 0 : (lon,lat) --> (x,y)
*             1 : (x,y) --> (lon,lat)
*
*****/
int orthproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct orth_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slon, slat, olon, olat, xo, yo;
    double        xn, yn, alon, alat, theta, a1, a2, ch, ra, ac;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slon = map.slون * DEGRAD;
        slat = map.slat * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        xo = re*cos(olat)*sin(olon-slon) - map.xo;
        ch = sin(slat)*sin(olat) + cos(slat)*cos(olat)*cos(olon-slon);
        if (ch < -0.00001) return -1;
        yo = re*(cos(slat)*sin(olat) - sin(slat)*cos(olat)*cos(olon-slon)) - map.yo;
        map.first = 1;
    }

    if (code == 0) {
        alat = (*lat)*DEGRAD;
        alon = (*lon)*DEGRAD - slon;
        ch = sin(slat)*sin(alat) + cos(slat)*cos(alat)*cos(alon);
        if (ch < -0.00001) return -1;
        *x = re*cos(alat)*sin(alon) - xo;
        *y = re*(cos(slat)*sin(alat) - sin(slat)*cos(alat)*cos(alon)) - yo;
    } else {
        xn = *x + xo;
        yn = *y + yo;
        ra = xn*xn + yn*yn;
        if (ra <= 0.0) {
            ac = 0.0;
            *lat = slat * RADDEG;
        } else {
            ra = sqrt(ra);
            ac = asin(ra/re);
            *lat = cos(ac)*sin(slat) + yn*sin(ac)*cos(slat)/ra;
            *lat = asin(*lat) * RADDEG;
        }
        a1 = xn*sin(ac);
        a2 = ra*cos(slat)*cos(ac) - yn*sin(slat)*sin(ac);
        if (fabs(a1) <= 0.0) {
            theta = 0.0;
        } else if (fabs(a2) <= 0.0) {
            theta = 0.5*PI;
            if (a2 < 0.0) theta = -theta;
        } else {
            theta = atan2(a1, a2);
        }
        *lon = (theta + slon) * RADDEG;
    }
    return 0;
}
```

Ⅲ. Azimuthal equidistant projection

```
#include <stdio.h>
#include <math.h>
#include "map_ini.h"

/*****
*
* [ Azimuthal Equidistant Projection ]
*
* o lon, lat : (longitude,latitude) at earth [degree]
* o x, y      : (x,y) coordinate in map [grid]
* o code = 0 : (lon,lat) --> (x,y)
*             1 : (x,y) --> (lon,lat)
* o map       : map parameter
*
*****/
int azedproj( lon, lat, x, y, code, map )

float *lon, *lat;          /* Longitude, Latitude [degree] */
float *x, *y;              /* Coordinate in Map [grid] */
int code;                  /* (0) lon,lat -> x,y (1) x,y -> lon,lat */
struct azed_parameter map;
{
    static double PI, DEGRAD, RADDEG;
    static double re, slon, slat, olon, olat, xo, yo;
    double alon, alat, ak, xn, yn, ra, theta, ac, a1, a2;

    if (map.first == 0) {
        PI = asin(1.0)*2.0;
        DEGRAD = PI/180.0;
        RADDEG = 180.0/PI;
        re = map.Re/map.grid;
        slon = map.slon * DEGRAD;
        slat = map.slat * DEGRAD;
        olon = map.olon * DEGRAD;
        olat = map.olat * DEGRAD;
        ak = sin(slat)*sin(olat) + cos(slat)*cos(olat)*cos(olon-slon);
        if (ak < -0.99999) return -1;
        if (ak > 0.99999) {
            ak = 1.0;
        } else {
            ak = acos(ak);
            ak = ak / sin(ak);
        }
        xo = re*ak*cos(olat)*sin(olon-slon) - map.xo;
        yo = re*ak*(cos(slat)*sin(olat) - sin(slat)*cos(olat)*cos(olon-slon)) - map.yo;
        map.first = 1;
    }

    if (code == 0) {
        alat = (*lat)*DEGRAD;
        alon = (*lon)*DEGRAD - slon;
        ak = sin(slat)*sin(alat) + cos(slat)*cos(alat)*cos(alon);
        if (ak < -0.99999) return -1;
        if (ak > 0.99999) {
            ak = 1.0;
        } else {
            ak = acos(ak);
            ak = ak / sin(ak);
        }
        *x = re*ak*cos(alat)*sin(alon) - xo;
        *y = re*ak*(cos(slat)*sin(alat) - sin(slat)*cos(alat)*cos(alon)) - yo;
    } else {
        xn = *x + xo;
        yn = *y + yo;
        ra = xn*xn + yn*yn;
        if (ra <= 0.0) {

```

```

        ac = 0.0;
        alat = slat;
    } else {
        ra = sqrt(ra);
        ac = ra/re;
        alat = cos(ac)*sin(slat) + yn*sin(ac)*cos(slat)/ra;
        alat = asin(alat);
    }
    *lat = alat*RADDEG;
    a1 = xn*sin(ac);
    a2 = ra*cos(slat)*cos(ac) - yn*sin(slat)*sin(ac);
    if (fabs(a1) <= 0.0) {
        theta = 0.0;
    } else if (fabs(a2) <= 0.0) {
        theta = PI*0.5;
        if (a2 < 0.0) theta = -theta;
    } else {
        theta = atan2(a1, a2);
    }
    *lon = (theta + slon)*RADDEG;
}
return 0;
}

```

※. header file

```

/*****
*
*   GPS80 Earth Reference Ellipsoid 의 경우
*
*       45N 부근에서의 지구반경은 6370.19584 km 이다.
*
*****/

struct albe_parameter {
    float Re;           /* 사용할 지구반경 [ km ] */
    float grid;         /* 격자간격 [ km ] */
    float slat1;        /* 표준위도 [degree] */
    float slat2;        /* 표준위도 [degree] */
    float olon;         /* 기준점의 경도 [degree] */
    float olat;         /* 기준점의 위도 [degree] */
    float xo;           /* 기준점의 X좌표 [격자거리] */
    float yo;           /* 기준점의 Y좌표 [격자거리] */
    int first;         /* 시작여부 (0 = 시작) */
};

struct azed_parameter {
    float Re;           /* 사용할 지구반경 [ km ] */
    float grid;         /* 격자간격 [ km ] */
    float slon;         /* 중심경도 [degree] */
    float slat;         /* 중심위도 [degree] */
    float olon;         /* 기준점의 경도 [degree] */
    float olat;         /* 기준점의 위도 [degree] */
    float xo;           /* 기준점의 X좌표 [격자거리] */
    float yo;           /* 기준점의 Y좌표 [격자거리] */
    int first;         /* 시작여부 (0 = 처음) */
};

struct eqdc_parameter {
    float Re;           /* 사용할 지구반경 [ km ] */
    float grid;         /* 격자간격 [ km ] */
    float slon;         /* 표준경도 [degree] */
    float slat;         /* 표준위도 [degree] */
    float olon;         /* 기준점의 경도 [degree] */
    float olat;         /* 기준점의 위도 [degree] */
    float xo;           /* 기준점의 X좌표 [격자거리] */
    float yo;           /* 기준점의 Y좌표 [격자거리] */
    int first;         /* 시작여부 (0 = 처음) */
};

struct lamc_parameter {
    float Re;           /* 사용할 지구반경 [ km ] */
    float grid;         /* 격자간격 [ km ] */
    float slat1;        /* 표준위도 [degree] */
    float slat2;        /* 표준위도 [degree] */
    float olon;         /* 기준점의 경도 [degree] */
    float olat;         /* 기준점의 위도 [degree] */
    float xo;           /* 기준점의 X좌표 [격자거리] */
    float yo;           /* 기준점의 Y좌표 [격자거리] */
    int first;         /* 시작여부 (0 = 시작) */
};

struct lame_parameter {
    float Re;           /* 사용할 지구반경 [ km ] */
    float grid;         /* 격자간격 [ km ] */
    float slon;         /* 표준경도 [degree] */
    float slat;         /* 표준위도 [degree] */
    float olon;         /* 기준점의 경도 [degree] */
    float olat;         /* 기준점의 위도 [degree] */
    float xo;           /* 기준점의 X좌표 [격자거리] */
    float yo;           /* 기준점의 Y좌표 [격자거리] */
    int first;         /* 시작여부 (0 = 처음) */
};

```

```

struct merc_parameter {
    float Re; /* 사용할 지구반경 [ km ] */
    float grid; /* 격자간격 [ km ] */
    float slon; /* 표준경도 [degree] */
    float slat; /* 표준위도 [degree] */
    float olon; /* 기준점의 경도 [degree] */
    float olat; /* 기준점의 위도 [degree] */
    float xo; /* 기준점의 X좌표 [격자거리] */
    float yo; /* 기준점의 Y좌표 [격자거리] */
    int first; /* 시작여부 (0 = 처음) */
};

struct milm_parameter {
    float Re; /* 사용할 지구반경 [ km ] */
    float grid; /* 격자간격 [ km ] */
    float slon; /* 표준경도 [degree] */
    float olon; /* 기준점의 경도 [degree] */
    float olat; /* 기준점의 위도 [degree] */
    float xo; /* 기준점의 X좌표 [격자거리] */
    float yo; /* 기준점의 Y좌표 [격자거리] */
    int first; /* 시작여부 (0 = 처음) */
};

struct oblm_parameter {
    float Re; /* 사용할 지구반경 [ km ] */
    float grid; /* 격자간격 [ km ] */
    float slon; /* 지도 극점 경도 [degree] */
    float slat; /* 지도 극점 위도 [degree] */
    float olon; /* 기준점의 경도 [degree] */
    float olat; /* 기준점의 위도 [degree] */
    float xo; /* 기준점의 X좌표 [격자거리] */
    float yo; /* 기준점의 Y좌표 [격자거리] */
    int first; /* 시작여부 (0 = 처음) */
};

struct orth_parameter {
    float Re; /* 사용할 지구반경 [ km ] */
    float grid; /* 격자간격 [ km ] */
    float slon; /* 표준경도 [degree] */
    float slat; /* 표준위도 [degree] */
    float olon; /* 기준점의 경도 [degree] */
    float olat; /* 기준점의 위도 [degree] */
    float xo; /* 기준점의 X좌표 [격자거리] */
    float yo; /* 기준점의 Y좌표 [격자거리] */
    int first; /* 시작여부 (0 = 처음) */
};

struct sinu_parameter {
    float Re; /* 사용할 지구반경 [ km ] */
    float grid; /* 격자간격 [ km ] */
    float slon; /* 표준경도 [degree] */
    float olon; /* 기준점의 경도 [degree] */
    float olat; /* 기준점의 위도 [degree] */
    float xo; /* 기준점의 X좌표 [격자거리] */
    float yo; /* 기준점의 Y좌표 [격자거리] */
    int first; /* 시작여부 (0 = 처음) */
};

struct ster_parameter {
    float Re; /* 사용할 지구반경 [ km ] */
    float grid; /* 격자간격 [ km ] */
    float slon; /* 중심경도 [degree] */
    float slat; /* 중심위도 [degree] */
    float olon; /* 기준점의 경도 [degree] */
    float olat; /* 기준점의 위도 [degree] */
    float xo; /* 기준점의 X좌표 [격자거리] */
    float yo; /* 기준점의 Y좌표 [격자거리] */
    int first; /* 시작여부 (0 = 처음) */
    int polar; /* 90N중심에 투영면이 60N을 지나는 경우 (1) */
};

```



```

struct trnm_parameter {
    float Re;          /* 사용할 지구반경 [ km ] */
    float grid;         /* 격자간격 [ km ] */
    float slon;         /* 원점의 경도 [degree] */
    float slat;         /* 원점의 위도 [degree] */
    float olon;         /* 기준점의 경도 [degree] */
    float olat;         /* 기준점의 위도 [degree] */
    float xo;           /* 기준점의 X좌표 [격자거리] */
    float yo;           /* 기준점의 Y좌표 [격자거리] */
    int first;         /* 시작여부 (0 = 처음) */
};

struct vand_parameter {
    float Re;          /* 사용할 지구반경 [ km ] */
    float grid;         /* 격자간격 [ km ] */
    float slon;         /* 표준경도 [degree] */
    float olon;         /* 기준점의 경도 [degree] */
    float olat;         /* 기준점의 위도 [degree] */
    float xo;           /* 기준점의 X좌표 [격자거리] */
    float yo;           /* 기준점의 Y좌표 [격자거리] */
    int first;         /* 시작여부 (0 = 처음) */
};

```

별첨

A. 구면삼각형에서의 삼각함수식

오른쪽 그림과 같은 구면삼각형의 경우
다음과 같은 법칙들이 성립한다.

Law of sines :

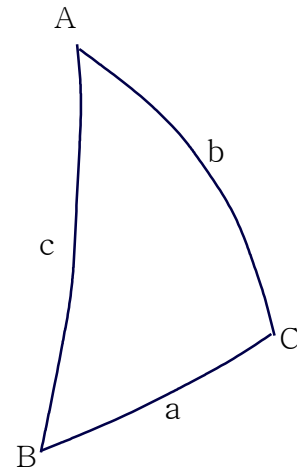
$$\frac{\sin a}{\sin A} = \frac{\sin b}{\sin B} = \frac{\sin c}{\sin C}$$

Law of cosines for sides :

$$\cos a = \cos b \cos c + \sin b \sin c \cos A$$

$$\cos b = \cos c \cos a + \sin c \sin a \cos B$$

$$\cos c = \cos a \cos b + \sin a \sin b \cos C$$



Law of cosines for angles :

$$\cos A = -\cos B \cos C + \sin B \sin C \cos a$$

$$\cos B = -\cos C \cos A + \sin C \sin A \cos b$$

$$\cos C = -\cos A \cos B + \sin A \sin B \cos c$$

그러면

$$\cos c = \sin \phi_1 \sin \phi + \cos \phi_1 \cos \phi \cos(\lambda - \lambda_0) \quad \leftarrow \text{Law of cosines for sides, 또는 벡터의 내적}$$

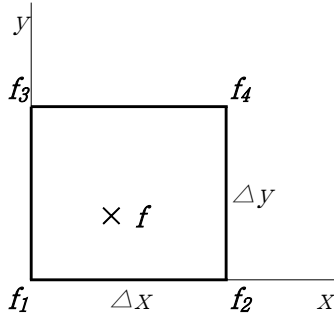
$$\sin c = \sqrt{\cos^2 \phi \sin^2(\lambda - \lambda_0) + (\cos \phi_1 \sin \phi - \sin \phi_1 \cos \phi \cos(\lambda - \lambda_0))^2} \quad \leftarrow \text{벡터의 외적}$$

$$\sin Az = \frac{\sin(\lambda - \lambda_0) \cos \phi}{\sin c} \quad \leftarrow \text{Law of sines}$$

$$\cos Az = \frac{\cos \phi_1 \sin \phi - \sin \phi_1 \cos \phi \cos(\lambda - \lambda_0)}{\sin c} \quad \leftarrow \text{Law of cosines for sides + Law of sines}$$

B. 격자자료의 변환

가. Linear Interpolation (선형내삽)



(f_1, f_2, f_3, f_4) 가 관측값이라 할 때

여기서 내부의 x 점에서의 값을 계산하라.

$f(x, y) = a + bx + cy + dxy$ 라 하면

$$f_1 = a$$

$$f_2 = a + b\Delta x = f_1 + b\Delta x$$

$$f_3 = a + c\Delta y = f_1 + c\Delta y$$

$$f_4 = a + b\Delta x + c\Delta y + d\Delta x\Delta y = f_2 + f_3 - f_1 + d\Delta x\Delta y$$

이 된다. 이 선형방정식을 풀면

$$f = f_1 + \frac{f_2 - f_1}{\Delta x} x + \frac{f_3 - f_1}{\Delta y} y + \frac{f_4 - f_2 - f_3 + f_1}{\Delta x \Delta y} xy$$

여기서 $x' = x/\Delta x$, $y' = y/\Delta y$ 라 하면

$$\therefore f = f_1 + (f_2 - f_1)x' + (f_3 - f_1)y' + (f_4 - f_2 - f_3 + f_1)x'y'$$

이것을 다르게 쓰면

$$f = f_1(1 - x' - y' + x'y') + f_2(x' - x'y') + f_3(y' - x'y') + f_4x'y'$$

위의 것을 다른 개념으로 계산할 수 있다.

f_2 와 f_1 사이를 선형 내삽하여 x 에서의 값을 구하면

$$g_1 = \frac{f_2 - f_1}{\Delta x} x + f_1$$

마찬가지로 f_4 와 f_3 사이의 선형내삽에서 x 위치에서의 값을 구하면

$$g_2 = \frac{f_4 - f_3}{\Delta y} y + f_3$$

그러면 (x, y) 에서의 값 f 는 g_1 와 g_2 의 y 방향으로 선형내삽에 의하여 구할 수 있다.

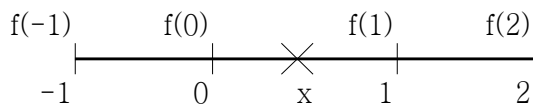
$$\therefore f = \frac{g_2 - g_1}{\Delta y} y + g_1$$

위 식을 대입하여 정리하면 앞과 같은 식이 된다. 즉

$$\begin{aligned} f &= y'((f_4 - f_3)x' + f_3 - (f_2 - f_1)x' - f_1) + (f_2 - f_1)x' + f_1 \\ &= f_1(1 - x' - y' + x'y') + f_2(x' - x'y') + f_3(y' - x'y') + f_4x'y' \end{aligned}$$

나. NMC Interpolation

● 1차원 내삽



옆에 그림처럼 $x = -1, 0, 1, 2$ 지점에 $f(-1), f(0), f(1), f(2)$ 값들이 있을 경우, $[-1, 2]$ 사이에 x 에서의 $f(x)$?

$f(x)$ 를 Taylor 전개하면

$$f(x) = f(0) + f'(0)x + f''(0)/2 \cdot x^2$$

따라서

$$f(-1) = f(0) - f'(0) + f''(0)/2 \quad \text{.....㉠}$$

$$f(0) = f(0) \quad \text{.....㉡}$$

$$f(1) = f(0) + f'(0) + f''(0)/2 \quad \text{.....㉢}$$

$$f(2) = f(0) + 2f'(0) + 2f''(0) \quad \text{.....㉣}$$

㉠ - 5 ㉡ + ㉣ 하면

$$f''(0) = -\frac{1}{4} (f(-1) + 3f(0) - 5f(1) + f(2))$$

㉠ - ㉡ + ㉢ 하면

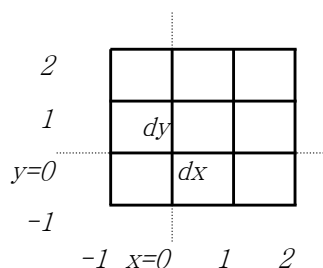
$$\frac{f''(0)}{2} = \frac{1}{4} (f(-1) - f(0) - f(1) + f(2))$$

$$\therefore f(x) = f(0) - \frac{1}{4} [f(-1) + 3f(0) - 5f(1) + f(2)] + \frac{x^2}{4} [f(-1) - f(0) - f(1) + f(2)]$$

위 식을 정리하면

$$\therefore f(x) = f(0) + x \left\{ f(1) - f(0) + \frac{(x-1)}{4} [f(-1) - f(0) - f(1) + f(2)] \right\}$$

● 2차원에서의 내삽



위 방법으로 $y = -1, 0, 1, 2$ 에서

x 축에 대하여 $x = dx$ 에서의 값들을 내삽하고

이것을 다시 y 축에 대하여 $y = dy$ 에서의 값을 내삽하여 구한다.

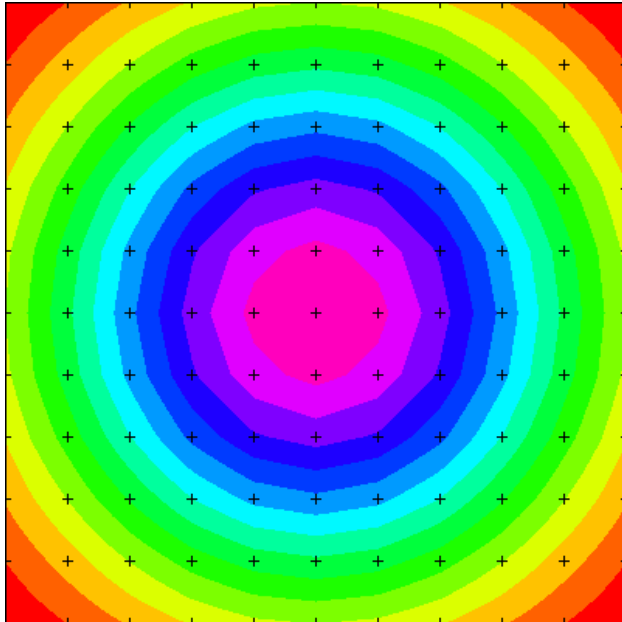
● 기울기

$$\frac{\partial f(x)}{\partial x} = f(1) - f(0) + \frac{(2x-1)}{4} [f(-1) - f(0) - f(1) + f(2)]$$

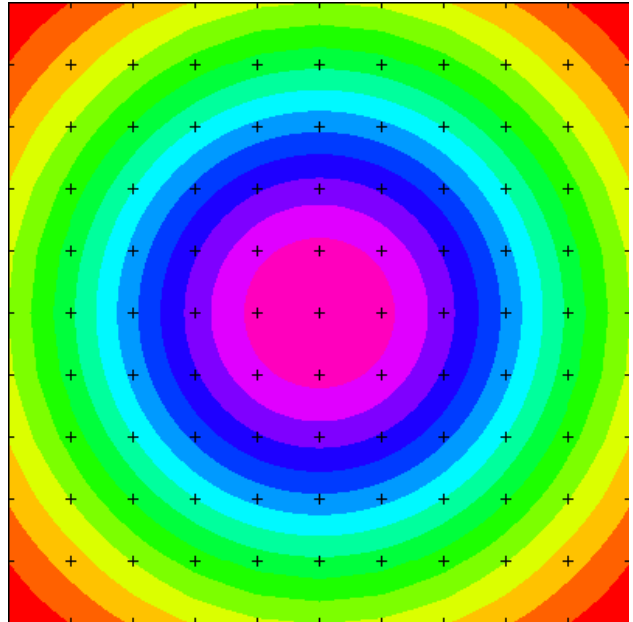
※ 이 방법은 NMC 격자자료와 위경도 자료 사이의 변환에 사용되던 것을 정리한 것임.

아래에서 [그림 B-1]은 종형의 격자자료에 대하여 선형내삽법을 사용하여 세분하여 그린 것이고, [그림 B-2]는 NMC방법을 사용하여 세분한 것이다. (그림에서 +는 격자점 위치이다)

[그림 B-1] 선형내삽



[그림 B-2] NMC 내삽



- 강수량이나 풍속 등과 같이 격자간격이 크고 자료의 기울기가 클수록 선형내삽의 경우 차이가 크다. 이러한 경우에 와도 분석과 같은 것을 하면 실제와 다른 결과가 나올수 있으니 주의할 필요가 있다.
- 그러나 위성이나 레이더 등과 같은 자료는 격자간격이 조밀하므로 큰 문제가 없다고 생각된다.