# 7.　cartopy の利用

現在、Basemap の開発は終了しているので、代替となる cartopy の利用を考えても良いでしょう。ここでは Basemap の代用として、似たような機能を提供している cartopy で作図する方法を紹介します。バージョン 0.17 と 0.18 で経度線や緯度線等の作図が大きく変更され、互換性のあるコードを作成することが難しくなったため、本稿はバージョン 0.18 以降を中心に解説しています。バージョン 0.17 での挙動については 7.5 節に記載しました。

## 7.1 cartopy の基本

### 7.1.1 地図を描く

cartopy は Basemap と似たように matlotlib 上で地図を表示するためのパッケージです。Basemap 同様に様々な図法に対応しており、等高線や陰影、矢印、矢羽等の作図を行うこともできます。まずは cartopy_sample.py を使い、正距円筒図法で作図を行なってみます（図 7－1－1）。

モジュールは import cartopy.crs as ccrs のようにインポートします。cartopy の場合も Basemap 同様、最初に matplotlib でプロット領域を作成するため、matplotlib もインポートします。サブプロットを生成する所では、matplotlib の fig.add_subplot を使います。これまで同様、最初の引数が縦に並べる数、2 つ目が横に並べる数、3 つ目がサブプロットのうちの何番目に当たるかを表します。ここでは、新たに projection オプションを使い、ccrs.PlateCarree()を与えることで、cartopy の正距円筒図法（6.2.1 節参照）を適用しています。ccrs.PlateCarree()のデフォルトでは東経 0 度が中央に来るようになっていますが、ここでは、central_longitude=180 を与えることで東経 180 度が中央に来るように変更しています。

```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs

fig = plt.figure() #  プロット領域の作成
# cartopy 呼び出し
ax=fig.add_subplot(1,1,1, projection=ccrs.PlateCarree(central_longitude=180))
```

ここまでの設定では図枠が表示されるだけなので、ax.メソッドのようなインスタンスメソッドを使い作図してみます。まずは ax.coastlines を使い海岸線を描きます。fig や ax の作成時、海岸線作図時のそれぞれの挙動が気になる場合には、ipython3 --pylab を起動して cartopy_sample.py の各行を順番に入力し、どのように表示されるか試しても良いでしょう。

```
ax.coastlines() # 海岸線を描く
```

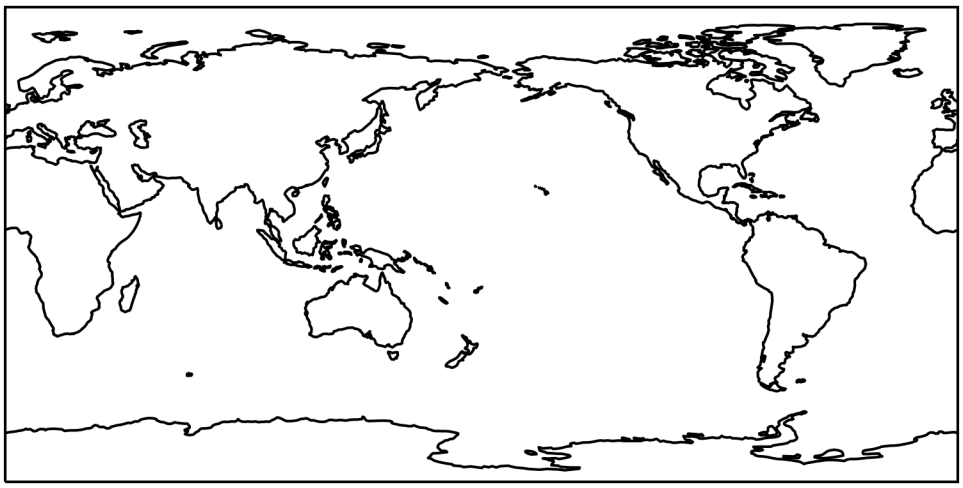

**図 7－1－1　cartopy で海岸線を描く**

なお、実行時に ShapelyDeprecationWarning が表示されるため、次のようなコードを追加して表示されないようにしています。この Warning は無視しておいて問題ないですが、将来の cartopy 更新時に修正されるものと思います。

```
import warnings
warnings.filterwarnings('ignore')
```

### 7.1.2 経度線と緯度線を描く

Basemap の時と同様、描いた図に経度線と緯度線を加えることも可能です（図７－１－２）。作図には cartopy_sample2.py を使いました。まず ax.gridlines で経度線・緯度線を描きます。正距円筒図法にするため、図法を指定する crs オプションに ccrs.PlateCarree()を与えます。ここで、経度線・緯度線の設定を行うために ax.gridlines の戻り値を gl に格納しています。経度線や緯度線を引く値は、gl.xlocator と gl.ylocator で与えます。その際に matplotlib の ticker に含まれる FixedLocator を利用しており、gl.xlocator=mticker.FixedLocator(**経度線を描く位置**)のように用います。こうすることで、指定した任意の位置に経度線を描くことができます。経度線を引く値 np.arange(0, 360, 30)は、開始点の０から終了点の 360 まで 30 毎の値を並べることを意味しています。緯度線についても同様に行います。

```
import matplotlib.ticker as mticker
gl = ax.gridlines(crs=ccrs.PlateCarree())
gl.xlocator = mticker.FixedLocator(np.arange(0, 360, 30)) #  経度線
gl.ylocator = mticker.FixedLocator(np.arange(-90, 90, 30)) #  緯度線
```

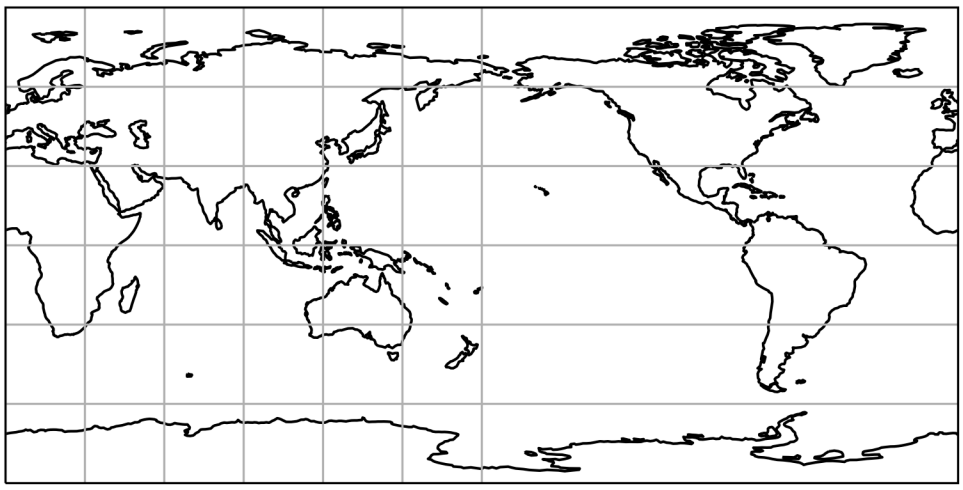

**図７－１－２　cartopy で経度線・緯度線を描く**

図７－１－２を見ると、西半球の経度線が描かれていないのが分かると思います。バージョン 0.18 では、経度線を引く値を np.arange(0, 360, 30)とすると、

180 より大きい値が認識されないようです。これを回避するために、np.arange(-180, 180, 30)として−180〜180 まで 30 毎の値を並べるようにしたものが、cartopy_sample3.py です。図７－１－３のように、西半球でも経度線が描かれるようになりました。

```
gl = ax.gridlines(crs=ccrs.PlateCarree())
gl.xlocator = mticker.FixedLocator(np.arange(-180, 180, 30))　# 経度線
gl.ylocator = mticker.FixedLocator(np.arange(-90, 90, 30))　# 緯度線
```

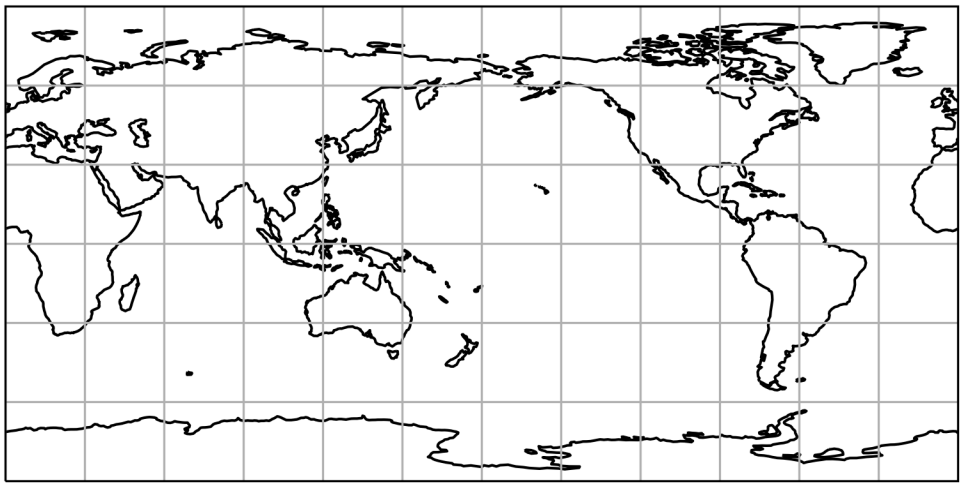

**図７－１－３　全ての経度線・緯度線が描かれるように修正した**

### 7.1.3 経度線と緯度線の見た目を調整する

次に、経度線・緯度線の見た目を変える方法を紹介します。図７－１－４のように経度線・緯度線を細い赤破線で描くプログラムが、cartopy_sample4.py です。線の幅は linewidth で設定し（デフォルト値：１）、細線にするため 0.5 にしました。線の色は color（図３－２－２、図３－２－３参照）、線種は linestyle です（表３－２－１参照）。それぞれ、赤色（'r'）、破線（'--'）に設定します。不透明度 alpha は 0.8 にしています。なお linewidth を lw、linestyle を ls、color を c とする省略形は、ここでは使うことができません。

```
gl = ax.gridlines(crs=ccrs.PlateCarree(), ¥
                  linewidth=0.5, linestyle='--', color='r', alpha=0.8)
```

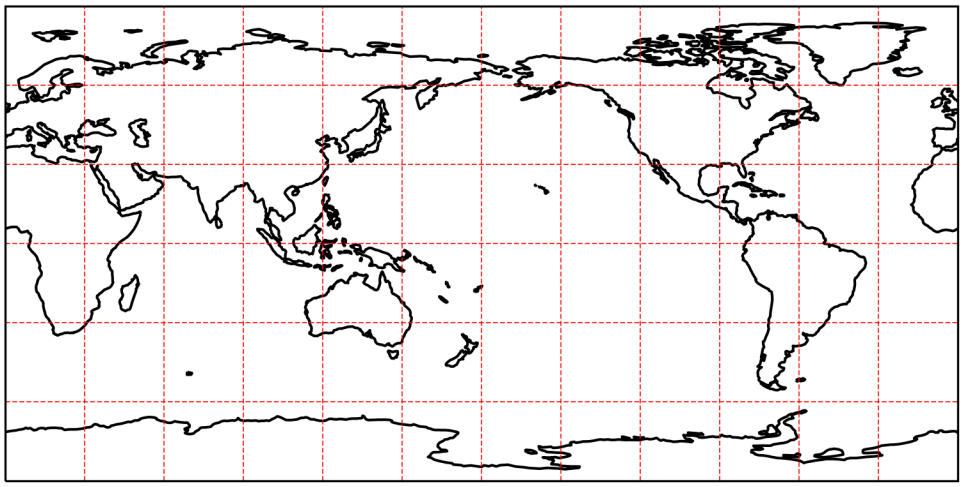

**図７－１－４　経度線・緯度線を赤破線で描く**

これらのオプションを使い linewidth=1、黒色（'r'）、点線（':'）に設定して、幅１の黒点線で経度線・緯度線を描きます（図７－１－５）。cartopy_sample5.py を使って作図しました。

```
gl = ax.gridlines(crs=ccrs.PlateCarree(), ¥
                  linewidth=1, linestyle=':', color='k', alpha=0.8)
```

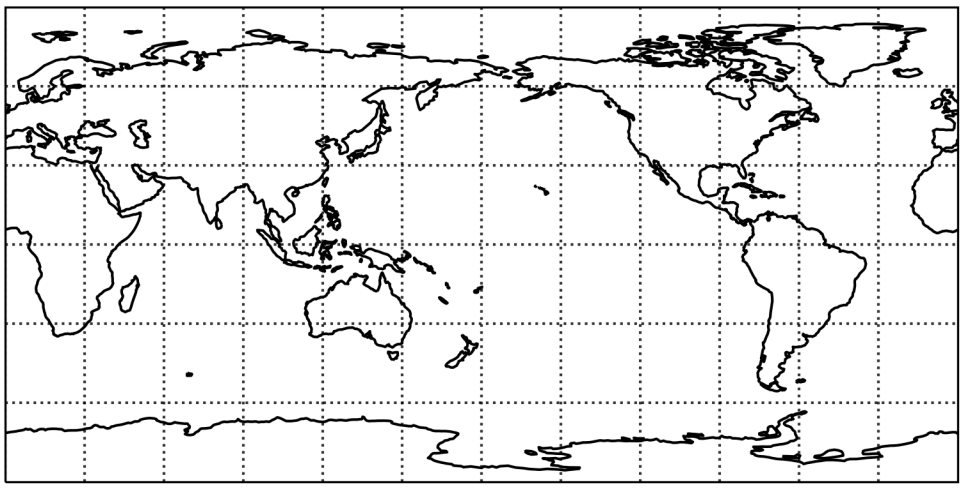

**図７－１－５　経度線・緯度線を黒点線で描く**

なお ax.coastlines にも線の色、線の幅、線種、不透明度を指定するオプションがあり、ax.coastlines(color='g')のように指定できます（緑色に変更する場合）。

### 7.1.4 経度線と緯度線にラベルを付ける

ax.gridlines には、経度線と緯度線にラベルを付ける draw_labels オプションがあります。draw_labels=True とすることでラベルが付きます（図７－１－６）。作図には cartopy_sample6.py を使いました。

```
gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True,
                  linewidth=1, linestyle=':', color='k', alpha=0.8)
```

同じスクリプトを使った場合でも、バージョン 0.18、0.19 と 0.20・0.21 で見た目が変わり、0.19 では経度方向のラベルの向きが、0.18 では緯度方向のラベルの有無が他と異なっています。

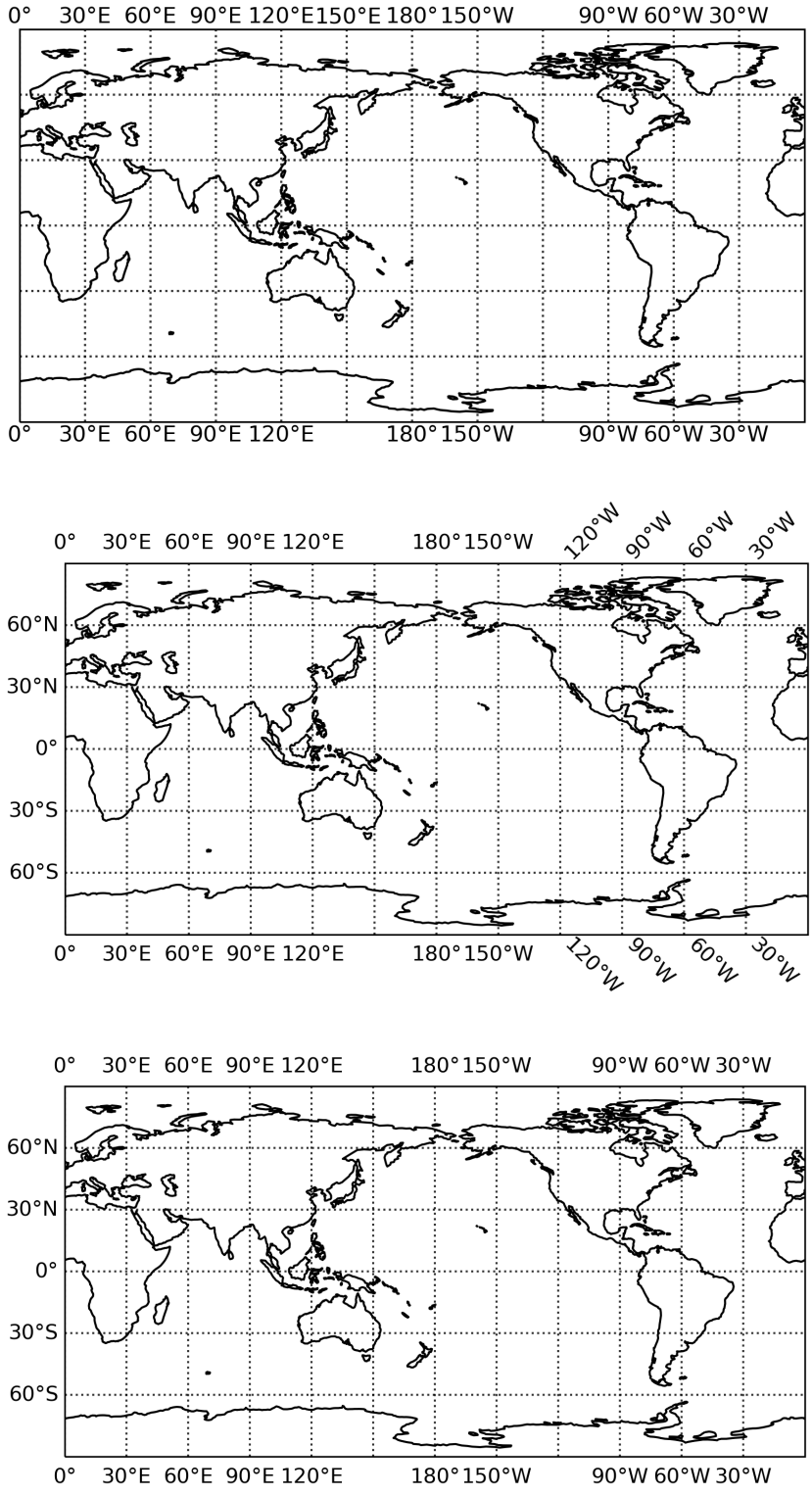


図７－１－６　経度線・緯度線ラベルを付ける。上はバージョン 0.18、中は 0.19、下は 0.20・0.21 の挙動

なお draw_labels オプションの挙動は、cartopy バージョン 0.17 と 0.18 の間で大きく変わりました。バージョン 0.17 では、うまくラベルの付く範囲が東経 0 度を中心とした場合に限られており、他の範囲では 7.1.5 節の方法でカスタマイズする必要がありました。

図 7－1－6 では、緯度線ラベルや経度線ラベルの一部が表示されていません。バージョン 0.19 では一部のラベルが斜めになっています。このように自動調整では不具合が出てしまいます。

図 7－1－7 は、図のサイズを大きくした場合の見た目で、全てのラベルが表示され、それなりの見た目になったように思えます。バージョン 0.18 と 0.19 で同じ見た目になります。作図には cartopy_sample7.py を使いました。

```
fig = plt.figure(figsize=(8, 6)) #  図のサイズを大きくする
```

上と右のラベルが表示されないように、次のような設定を加えました。右のラベルはここでは表示されていないため、上のラベルだけが消えました。

```
gl.top_labels = False #  上のラベルを消す
gl.right_labels = False #  右のラベルを消す
```

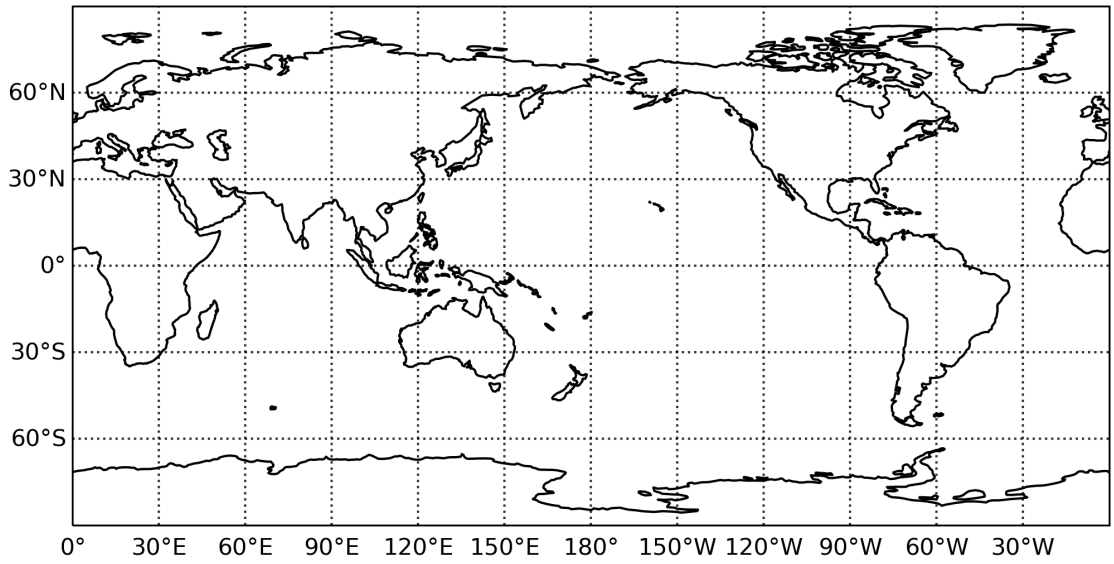


**図 7－1－7　図のサイズを大きくすると経度線・緯度線ラベルが全て表示される**

なお同様のオプションが、下のラベル（gl.bottom_labels）や左のラベル（gl.left_labels）にも設定可能です。

図の中心が東経0度の場合も試してみましょう（cartopy_sample8.py）。図7－1－8では、西経180度が左端、東経180度が右端にきました。左端の180°Wが180°のように表示され右端のラベルが表示されないため、次のようにラベルを描く範囲を150°Wから150°Eまでに変更しています。

```
gl.xlocator = mticker.FixedLocator(np.arange(-150, 180, 30))   # 経度線
```

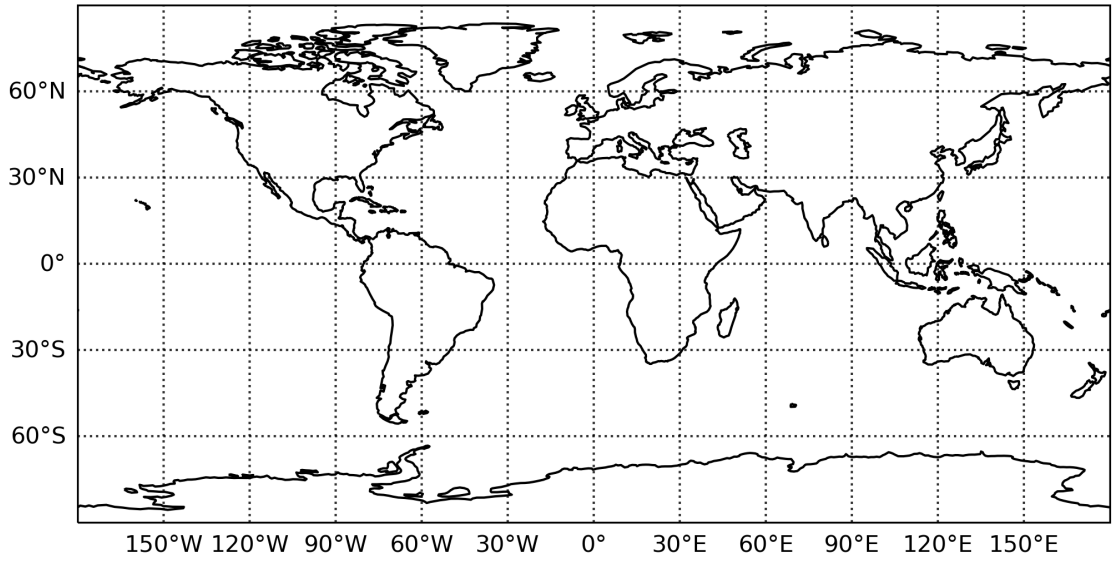


**図7－1－8　東経0度を中心とした場合**

### 7.1.5 経度線、緯度線ラベルのカスタマイズ

ここまでは、ax.gridlines のオプションを使った方法を紹介しましたが、カスタマイズが行いやすい方法として、matlotlib でラベルを描く ax.set_xticks、ax.set_yticks を使う方法も紹介しておきます。この方法では、経度線と緯度線は ax.gridlines で描き、ラベルはこれとは別に ax.set_xticks、ax.set_yticks で描きます。まずは、経度線と緯度線を描く値とラベルを描く値を同じに設定します。共通した値を使うので、xticks、yticks という配列を作っておき、それぞれの値を格納しておきます。dlon、dlat は、それぞれ経度線、緯度線ラベルを描く間隔を表しています。緯度線ラベルを 90°S から 90°N まで 30°毎に付けるため、np.arange(-90, 90.1, dlat)としました。作図には cartopy_sample9.py を使いました。

```
# 緯度線、経度線とラベルを描く値
dlon, dlat = 30, 30
xticks = np.arange(-180, 180, dlon)
yticks = np.arange(-90, 90.1, dlat)
```

まず ax.gridlines で、これまで同様に経度線と緯度線を描きます。

```
gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=False,
                  linewidth=1, linestyle=':', color='k', alpha=0.8)
gl.xlocator = mticker.FixedLocator(xticks) #  経度線を描く値
gl.ylocator = mticker.FixedLocator(yticks) #  緯度線を描く値
```

次に、ax.set_xticks、ax.set_yticks を使い、経度線と緯度線ラベルを描く値を指定します。オプションとして crs を与えることができ、ccrs.PlateCarree()を指定します。図 7－1－8 のように、180 度を中心として東経側にも西経側にもラベルが描かれるようになりましたが、経度表記に直したくなります。

```
ax.set_xticks(xticks, crs = ccrs.PlateCarree()) #  経度線ラベルの値
ax.set_yticks(yticks, crs = ccrs.PlateCarree()) #  緯度線ラベルの値
```

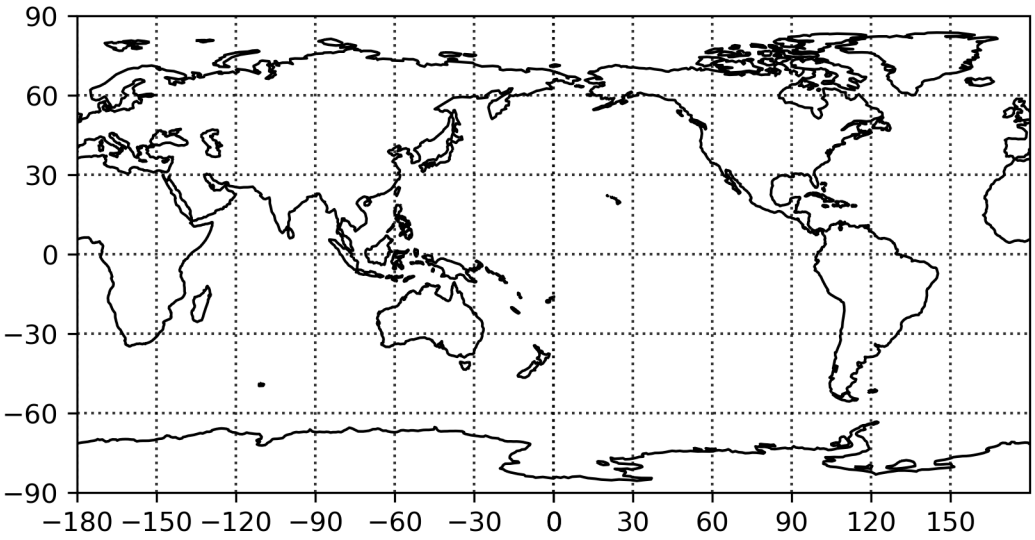


**図７－１－９　ax.set_xticks、ax.set_yticks を使い、経度線と緯度線ラベルを描いた**

cartopy_sample10.py は、経度線・緯度線ラベルを度数表記にするためのプログラムです。経度線ラベルの表記は ax.xaxis.set_major_formatter で変更します。その際に、cartopy.mpl.ticker に含まれる LongitudeFormatter を使って書式変更します。デフォルトのままでは東経０度が「０°E」ではなく「０°」と表示されてしまうため、オプションとして zero_direction_label=True を与えました。図７－１－10 のように、経度方向のラベルが 180 度を中心とした数字から東経（E)、西経（W）を区別した度数表記に変わりました。緯度線ラベルについても同様に、ax.yaxis.set_major_formatter の書式を変更します。緯度の場合は LatitudeFormatter ですが、赤道は「０°」表記で良いためデフォルトのまま使用しました。

```
from cartopy.mpl.ticker import LongitudeFormatter, LatitudeFormatter
…
#  目盛りの表示形式を度数表記にする
ax.xaxis.set_major_formatter(LongitudeFormatter(zero_direction_label=True))
ax.yaxis.set_major_formatter(LatitudeFormatter())
```

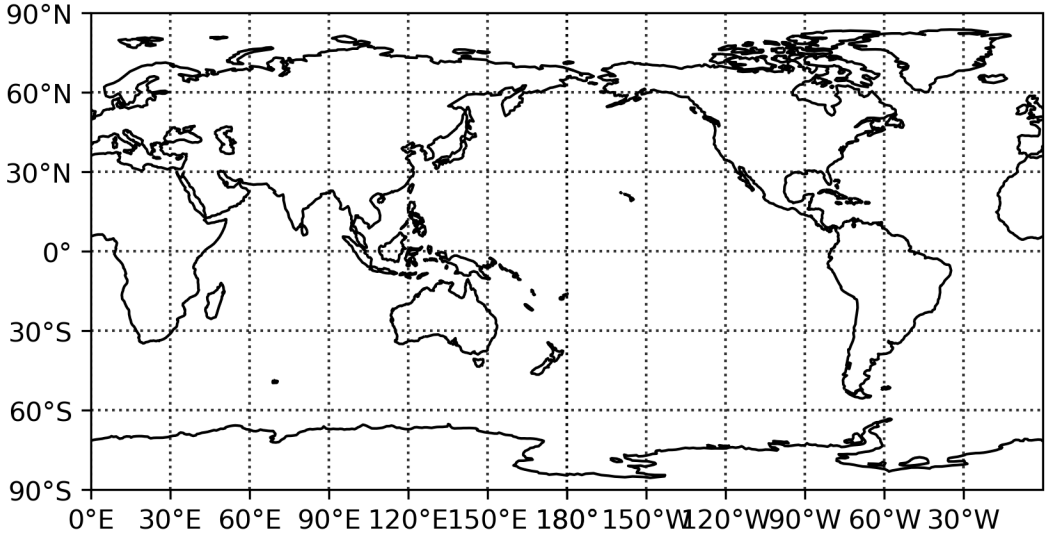


**図７－１－10　度数表記に変える**

7.1.6 経度線の間隔と経度線ラベルの間隔を別々にする

図７－１－10 では、150°W と 120°W のように一部のラベルが重なっています。ラベルの文字は少し小さいように思えますが、これ以上拡大すると隣のラベルと判別できなくなってしまうので、経度線は 30 度毎のまま、経度線ラベルだけ 60 度毎に描いてみます。経度線・緯度線を描く部分は変更ありませんが、経度線・緯度線ラベルを描くために、新たに dlon_lab、dlat_lab で経度線、緯度線ラベルの間隔を指定し、xticks_lab、yticks_lab という配列を作って、経度線、緯度線ラベルを描く値を格納しておきます。経度線ラベルの間隔のみ 60 度にしておくことで、経度線の間隔は 30 度のままラベルの間隔を変更できます。作図には cartopy_sample11.py を用いました。

```
dlon_lab, dlat_lab = 60, 30 #  経度線・緯度線の間隔
xticks_lab = np.arange(-180, 180, dlon_lab) #  経度線ラベルを描く値
yticks_lab = np.arange(-90, 90.1, dlat_lab) #  緯度線ラベルを描く値
```

経度線ラベル、緯度線ラベルを描く部分では、これら xticks_lab、yticks_lab を使います。

```
ax.set_xticks(xticks_lab, crs=ccrs.PlateCarree()) #  経度線ラベル
ax.set_yticks(yticks_lab, crs=ccrs.PlateCarree()) #  緯度線ラベル
```

目盛り線ラベルのサイズは、ax.axes.tick_params で設定します。オプションとして labelsize=12 とすれば、文字サイズが 12 ポイントに設定されます。

```
ax.axes.tick_params(labelsize=12) #  目盛り線ラベルのサイズを指定
```

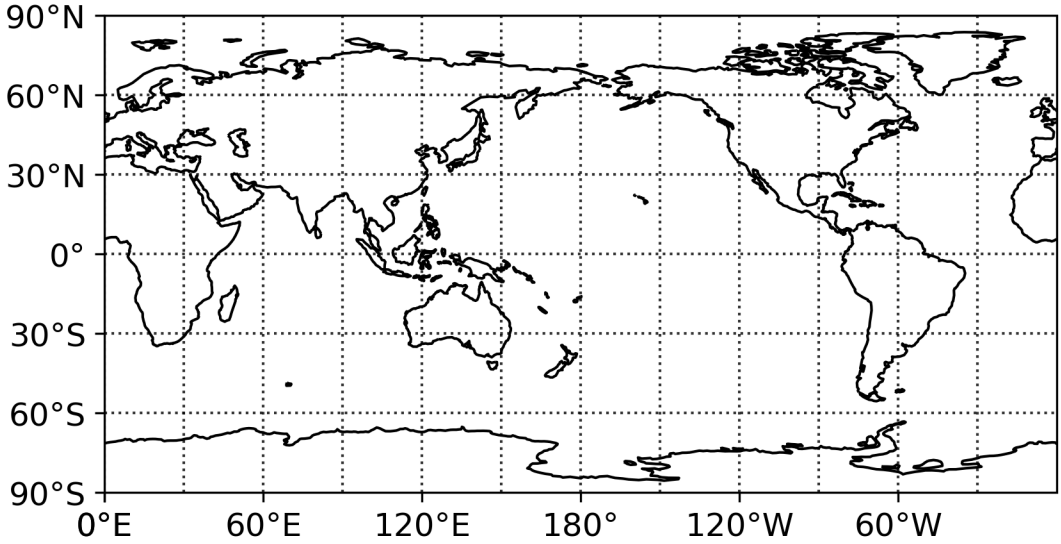


**図 7－1－11　経度線を 30 度毎、経度線ラベルを 60 度毎に描く**

ax.axes.tick_params では、他にも多くの項目を変更可能です（表 7－1－1、図 7－1－12）。オプションのうち、axis と which は変更を x 軸と y 軸のどちらに適用するや主目盛り、副目盛りのどちらに適用するかを指定するものです。指定がない場合は全てに同じ変更が適用されます。

目盛り線の設定は、幅を width（デフォルト：1）、長さを length（デフォルト：3.5）、色を color（デフォルト：黒）で変更できます。目盛り線の向きは、デフォルトでは枠線の外側（direction='out'）ですが、内側に変更することもでき、その場合には direction='in'とします。

目盛り線ラベルは、目盛り線とは別に設定するようになっており、色を labelcolor（デフォルト：黒）、ラベルを回転させる角度を labelrotation（デフ

ォルト：０）で変更します。

表７－１－１　matplotlib の axes.axes.tick_params で使用可能なオプション一覧

| オプション | 説明 |
|---|---|
| width | 目盛り線の幅、デフォルト：1 |
| length | 目盛り線の長さ、デフォルト：3.5 |
| color | 目盛り線の色、デフォルト：'k'（黒） |
| direction | 目盛り線の向き、'in', 'out', 'inout'、デフォルト：'out'（外側） |
| bottom, top, left, right | 下、上、左、右の目盛り線を描くかどうか、デフォルト；bottomとleftがTrue |
| pad | 目盛り線とラベルの間隔、デフォルト：4 |
| labelsize | 目盛り線ラベルの文字サイズ、<br>数字か次の文字列：'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large' |
| labelcolor | 目盛り線ラベルの色、デフォルト：'k'（黒） |
| labelrotation | 目盛り線ラベルを回転させる角度、デフォルト：0 |
| labelbottom, labeltop,<br>labelleft, labelright | 下、上、左、右の目盛り線ラベルを描くかどうか、<br>デフォルト；bottomとleftがTrue |
| axis | x軸、y軸のどちらに適用するか、'x', 'y', 'both'、デフォルト値：'both'（x、y軸両方） |
| which | 主、副どちらに適用するか、'major', 'minor', 'both'、デフォルト値：'both'（両方） |
| zorder | 目盛り線とラベルが描かれる順番 |

ax.axes.tick_params(オプション)：変更したい項目を指定

ax.axes.tick_params(オプション, axis='x')：x 軸で変更したい項目を指定

ax.axes.tick_params(オプション, axis='y')：y 軸で変更したい項目を指定

ax.axes.tick_params(オプション, axis='x', which='major')：x 軸の主目盛りについて、変更したい項目を指定

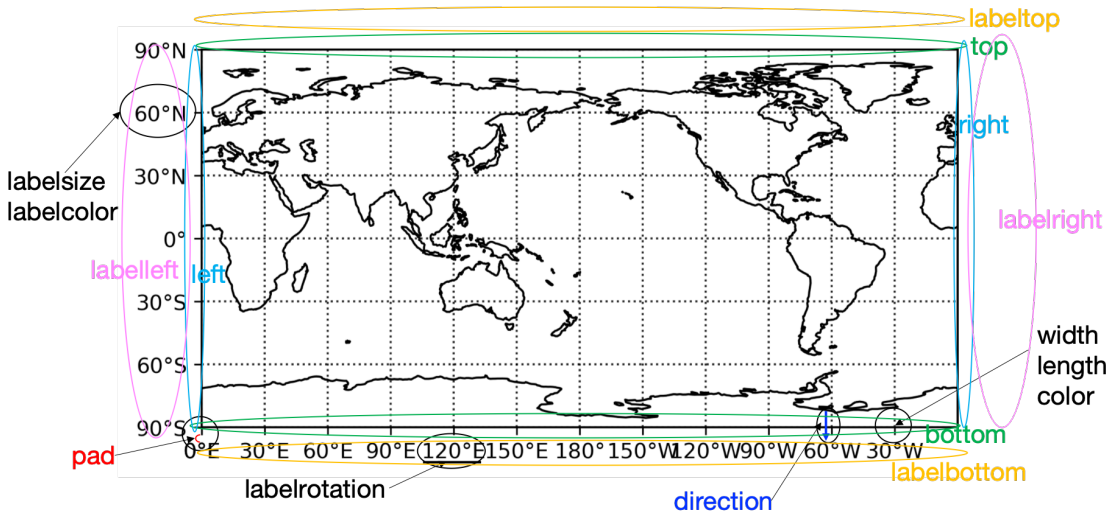


図７－１－12　ax.axes.tick_params で変更される箇所

図枠の下側、上側、左側、右側に目盛り線を付けるかどうかを決める bottom、top、left、right オプションがあり、デフォルトでは下側と左側だけに目盛り線が付きます。目盛り線ラベルを描くかどうかは、labelbottom、labeltop、labelleft、labelright オプションで設定し、こちらもデフォルトでは下側と左側だけに付くようになっています。

これらのオプションを使い、図枠の上側と右側にも目盛り線と目盛り線ラベルを付けてみます（図７－１－13）。作図には、cartopy_sample12.py を使いました。上側と右側に目盛り線を付けるため、top=True、right=True としています。また上側と右側の目盛り線ラベルは、labeltop=True、labelright=True で付けます。

```
ax.axes.tick_params(labelsize=12,
                    top=True, right=True, labeltop=True, labelright=True)
```

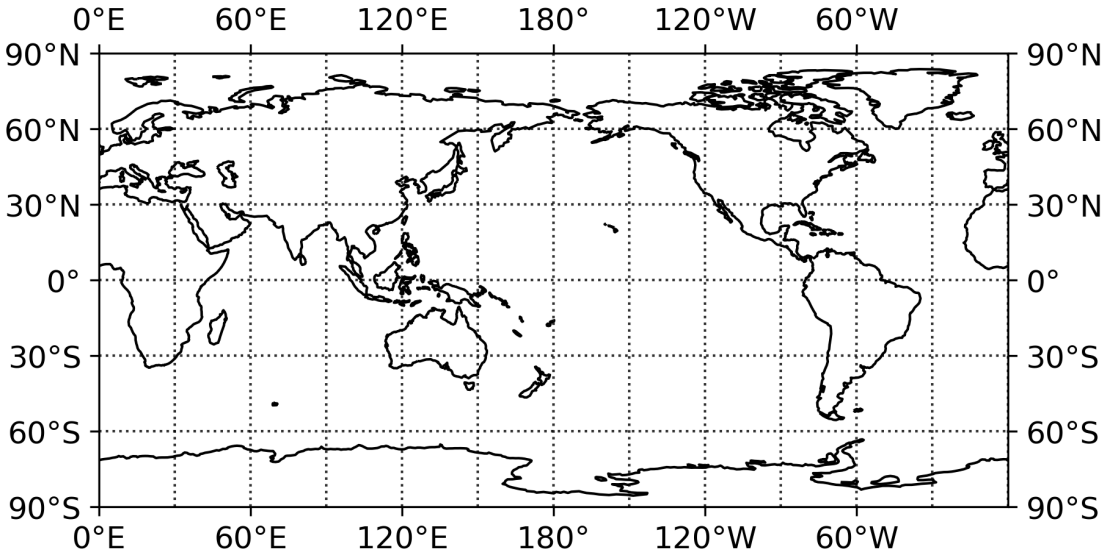


図７－１－13　図枠の上側と右側にも目盛り線と目盛り線ラベルを付ける

### 7.1.7 陸と海を塗り分けて描く

cartopy でも Basemap 同様に陸と海を塗り分けるツールが提供されています。ここでは、cartopy.feature を使って塗り分けてみます（図 7－1－14）。作図には cartopy_sample13.py を使います。ここでは、変更される部分が分かりやすくなるように、プログラム中の ax.coastlines()を無効にしているため、海岸線は描かれていません。cartopy.feature を使って作図する際には、初回のみデータのダウンロードが行われるため、ネットワークに接続されていない環境では cartopy_sample13.py でエラーが発生します。

最初に、作図に必要な cartopy.feature を import して cfeature として利用します。ax.add_feature(cfeature.**塗り潰す対象**)で色を付けることができ、陸に色を付ける場合は cfeature.LAND、海に色を付ける場合が cfeature.OCEAN です。なお cfeature.OCEAN では、海洋に加えて湖のうちカスピ海だけは塗り潰されています。

```
import cartopy.feature as cfeature
ax.add_feature(cfeature.LAND) #  陸に色を付ける
ax.add_feature(cfeature.OCEAN) #  海に色を付ける
```

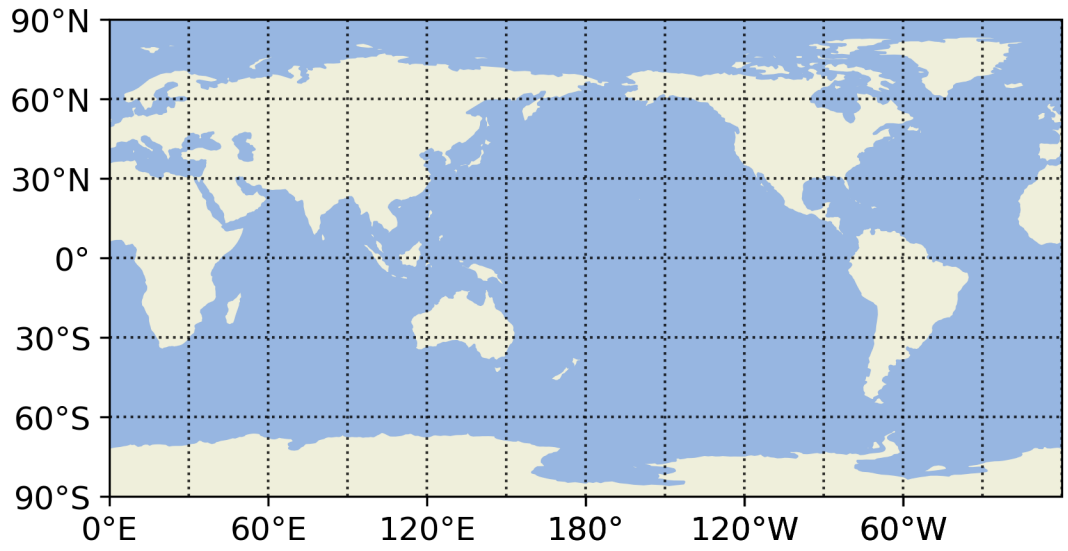


**図 7－1－14　陸と海を塗り分ける**

### 7.1.8 海岸線と湖、川を描く

さらに、海岸線と湖、川を追加していきます（図７－１－15）。海岸線を追加するには cfeature.COASTLINE、湖を描くには cfeature.LAKES、川を描くには cfeature.RIVERS を使います。黒実線の海岸線に加え、五大湖やバイカル湖、アマゾン川、ナイル川など主要な湖や河川が水色で表示されるようになりました。作図には、cartopy_sample14.py を使いました。

なお海岸線を描く場合、デフォルトでは ax.coastlines よりは太い線になります。線の太さは linewidth オプションを使うことにより変更可能で、少し細い線にするため linewidth=0.8 としています。他には不透明度を表すオプション alpha（0～１の範囲、１で不透明、０で透明）も利用可能です。

```
ax.add_feature(cfeature.COASTLINE, linewidth=0.8) #  海岸線を描く
ax.add_feature(cfeature.LAKES) #  湖を描く
ax.add_feature(cfeature.RIVERS) #  川を描く
```

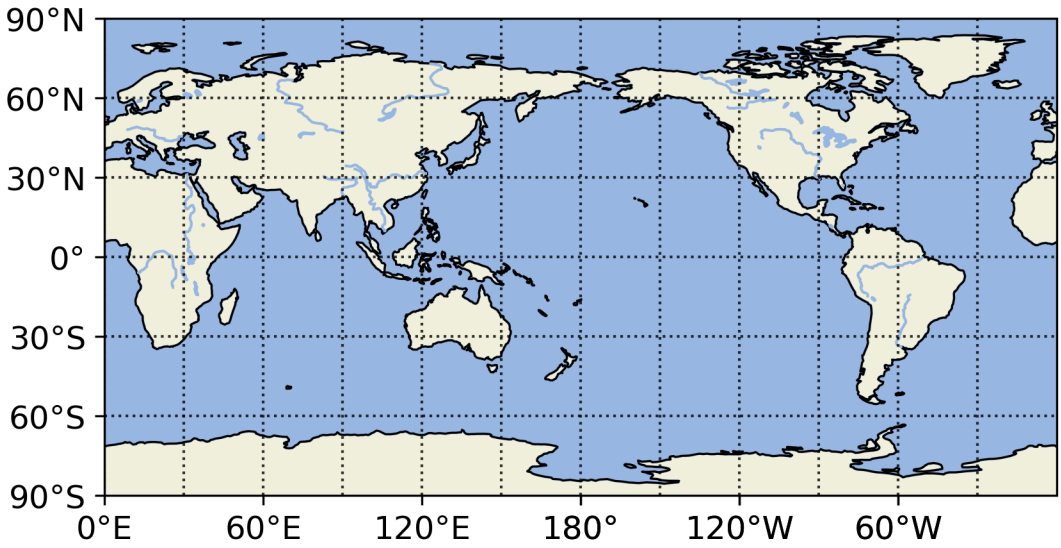


**図７－１－15　海岸線と湖、川を描く**

7.1.9 陸上に国境線を描く

陸上に国境線を描くこともでき、cfeature.BORDERS を使います。点線で国境線が描かれるようになりました（図７－１－16）。破線にするためのオプションとして linestyle='--'、線の幅を細くするため linewidth=0.8 を加えています。作図には、cartopy_sample15.py を使いました。

```
ax.add_feature(cfeature.BORDERS, linestyle='--', linewidth=0.8) #  国境線
```

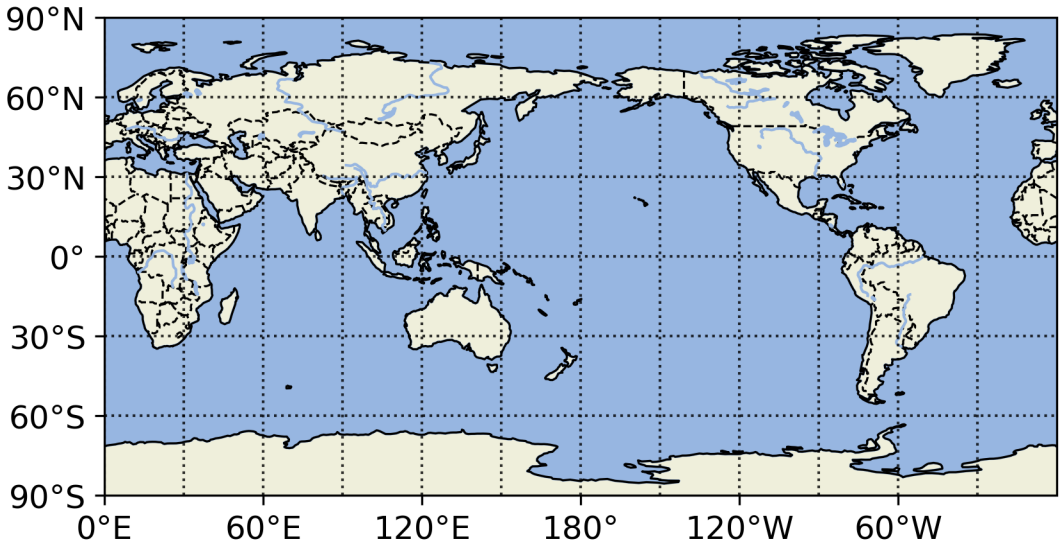


**図７－１－16　国境線を描く**

7.1.10 陸や海、湖を指定した色で塗り潰す

cartopy.feature はデフォルトでも綺麗に陸と海が塗り潰されているので、変更する必要はないかもしれませんが、色を指定して塗り潰すことも可能になっています。その際には、オプションとして color を使います。ここでは、Basemap の時に陸と海の塗り分けを行った図６－１－９と同じ色を付けてみます（図７－１－17）。陸を緑色にするために color='g'、海と湖を水色にするために color='aqua'としています。湖の周りが黒線にならない違いはありますが、図６－１－９と似た図になりました。作図には、cartopy_sample15.py を使いました。

```
ax.add_feature(cfeature.LAND, color='g') #  陸を緑色で塗り潰す
ax.add_feature(cfeature.OCEAN, color='aqua') #  海を水色で塗り潰す
ax.add_feature(cfeature.LAKES, color='aqua') #  湖を水色で塗り潰す
```

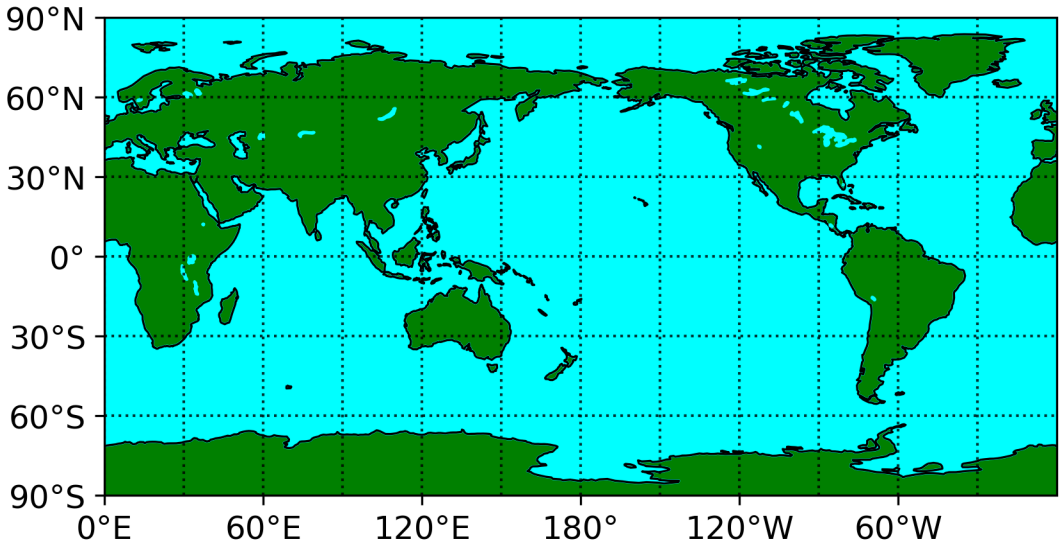


**図７－１－17　陸を緑色、海と湖を水色で塗り潰す**

なお、川と海岸線、国境線を描く所でも color オプションは有効ですが、意図していない部分まで塗り潰されてしまうので、使わない方が良いでしょう。

## 7.2 cartopy を使った等高線、陰影の作図

ここでは、Basemap の所で行ったように、気象データの形式の 1 つである NetCDF データを読み込み、作図を行います。気象データの形式や読み込みについては、6.4 節を参照してください。6.5.3 節で行った時と同様に、7 月の平均値を等高線、2018 年 7 月の海面更正気圧（SLP）偏差を陰影で作図できるようにしていきます。作図には、月平均した SLP データの NetCDF ファイル（slp.mon.mean.nc）が必要なので、まだダウンロードしていない場合には、basemap_nc2bin.py でダウンロードしておきます。

### 7.2.1 等高線を描く

まずは SLP データを利用して等高線を作成します（図 7－2－1）。作図には cartopy_contour.py を使いました。basemap_contour6.py で図 6－5－6 を作図した時と同様に、NCEP 再解析データの NetCDF ファイルを読み込み、取り出した 1 次元の経度データを lon、1 次元の緯度データを lat、空間 2 次元、時間 1 次元の SLP データを slp という変数に格納しています。

```
file_name = "slp.mon.mean.nc"
nc = netCDF4.Dataset(file_name, 'r') # NetCDF データの読み込み
lon = nc.variables["lon"][:] #  経度データ
lat = nc.variables["lat"][:] #  緯度データ
slp = nc.variables["slp"][:] # SLP データ
nc.close() #  ファイルを閉じる
```

次も同様に、7 月の SLP 平均値を計算して slp_jul に格納します。時間平均を行なったので、slp_jul は空間 2 次元のデータです。プログラム中では、作成したデータのサイズを numpy 配列名.shape を使って書き出しており、lon: (144,)、lat: (73,)、slp: (860, 73, 144)、slp_jul: (73, 144)のように、lon と lat は 1 次元、slp は 3 次元、slp_jul は 2 次元のデータとなっていることが分かります。このメソッドでは、配列の形状をタプルで書き出すので、1 次元の場合でもカンマが付いています。

```
tstr = (1981 - 1948) * 12 + 6 #  開始点：1981 年 7 月
tend = (2010 - 1948 + 1) * 12 #  終了点：2010 年の 12 月
slp_jul = slp[tstr:tend:12, :, :].mean(axis=0) # 7 月の SLP 平年値
```

cartopy で等高線を描くコマンドは、ax.contour(**引数 1**, **引数２**, **引数３**)です。5.2 節で紹介した matplotlib の plt.contour と同じですが、ax を定義した際に cartopy の正距円筒図法を適用したので、正距円筒図法上で等高線を描くようになっています。5.2 節の場合と同様、ax.contour の 1 番目の引数は x 軸上の位置、２番目の引数は y 軸上の位置で、３番目の引数は等高線を描くためのｚ軸データに当たります。z 軸データとしては、2 次元の slp_jul を用います。x, y 軸データとしては、１次元の経度・緯度データ lon、lat から np.meshgrid を使って作成した２次元データ x、y を使います。x、y も形状を書き出しており、x: (73, 144)、y: (73, 144)のように２次元であることが確認できます。４番目の引数として、等高線を描く値を与えることができ、その clevs は、5.2 節のときと同様に 4 hPa 毎に設定しています。４番目の引数を与える代わりに levels=clevs としても同じです。

```
x, y = np.meshgrid(lon-180.0, lat) #  経度、緯度データ
clevs = np.arange(np.floor(slp_jul.min() - np.fmod(slp_jul.min(), 20)), ¥
                        np.ceil(slp_jul.max()) + 1, 4)
ax.contour(x, y, slp_jul, clevs, linewidths=0.8, colors='k')
```

等高線を描く際には、5.2 節で紹介した plt.contour のオプションが利用できます（表５－２－１）。ここでは、線の幅を linewidths=0.8（デフォルト１）、線の色を colors='k'（黒色）に設定しました。

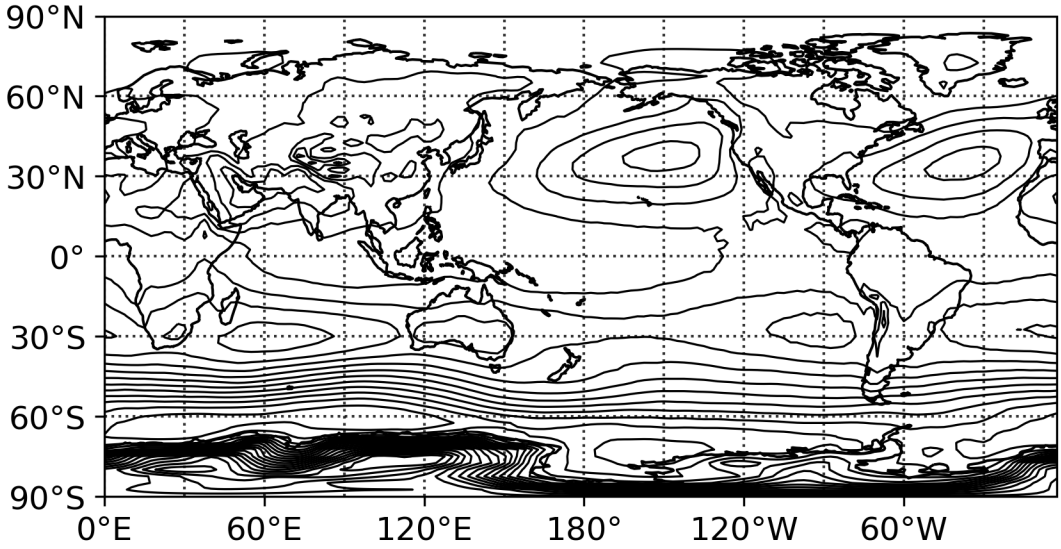


**図７－２－１　NCEP 再解析データの 1981 年から 2010 年までの 30 年間の 7 月 SLP を平均して等高線を描いた**

7.2.2 等高線ラベルを描く

図６－５－６のような等高線になりましたが、等高線に気圧を示すラベルが付いていません。ラベルを付けるには、Basemap で等高線ラベルを付けた時と同様に、等高線を描いた値 cs.levels を clevels に格納し、ax.contour の戻り値 cs にある、cs.clabel に clevels を渡す方法を使います（cartopy_contour2.py)。ここでは、clevels[::5]を与えて５飛ばし(4 hPa 毎に等高線を描いたので 20 hPa 毎)に等高線ラベルを描いています(図７－２－２)。ラベルの付く場所は、cartopy のバージョンによって変わります。ここではバージョン 0.20 のものを用いています、

```
cs = ax.contour(x, y, slp_jul, clevs, linewidths=0.8, colors='k') #  等高線
clevels=cs.levels #  ラベルを付ける値
cs.clabel(clevels[::5], fontsize=12, fmt="%d") #  等高線ラベル
```

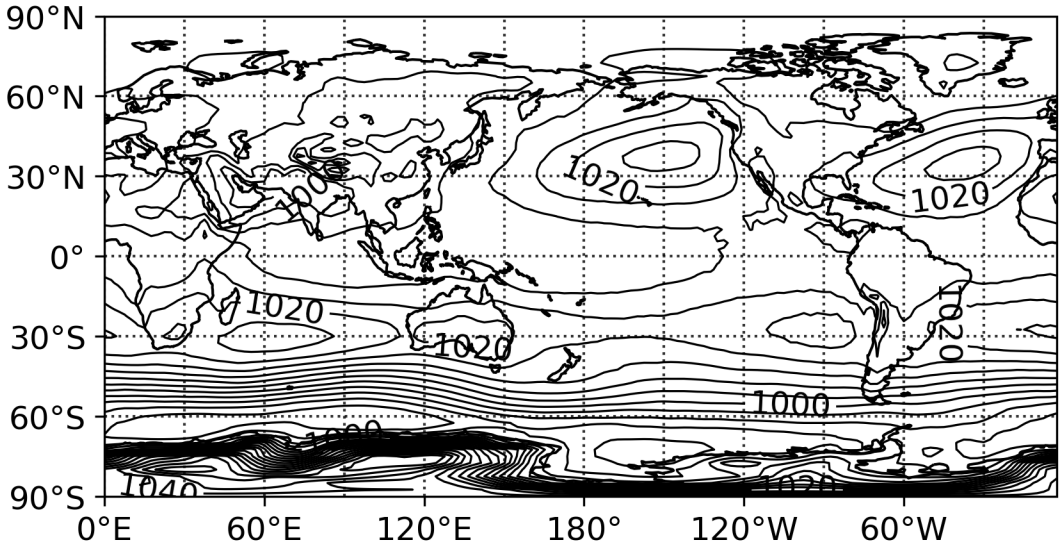


**図７－２－２　等高線ラベルを描いた**

7.2.3 陰影を描く

それでは、2018 年 7 月の SLP 偏差を陰影で描きます（図７－２－３）。図６－５－７とほとんど同じ図になりました。作図には cartopy_contourf.py を用いました。6.5.3 節同様に 2018 年 7 月の偏差を計算し、slp_anom に格納しておきます。陰影の範囲の決め方も同様に、正が赤、負が青で領域の範囲の中央が 0 になるようにしており、clevs の範囲の両端は、偏差の絶対値の最大値よりも 1 大きい値にしています。

陰影を描くコマンドは、ax.contourf(**引数 1**, **引数２**, **引数３**)です。matplotlib の plt.contourf と同じなので、5.3 節で紹介した plt.contourf のオプションが利用できます（表５－３－２）。1 番目から 3 番目までの引数は等高線を描く ax.contour と同じで、x 軸上の位置、y 軸上の位置、z 軸データを与えます。4 番目の引数には陰影を描く値 clevs を与えていますが、等高線の場合と同様、引数を与える代わりに levels=clevs とも記述できます。陰影を描く色は cmap オプションで色テーブルの名前から設定することができ、cmap='bwr'（青～白～赤と変化）としました。色テーブルの名前は、図４－５－７にあるものが利用できます。

```
n = (2018 - 1948) * 12 + 6
slp_anom = slp[n, :, :] - slp_jul # 2018 年 7 月の偏差
work = max(np.abs(np.floor(slp_anom.min())), np.abs(np.ceil(slp_anom.max())))
clevs = np.arange(-(work+1), work + 1) #  陰影を描く値
ax.contourf(x, y, slp_anom, clevs, cmap="bwr") #  陰影を描く
```

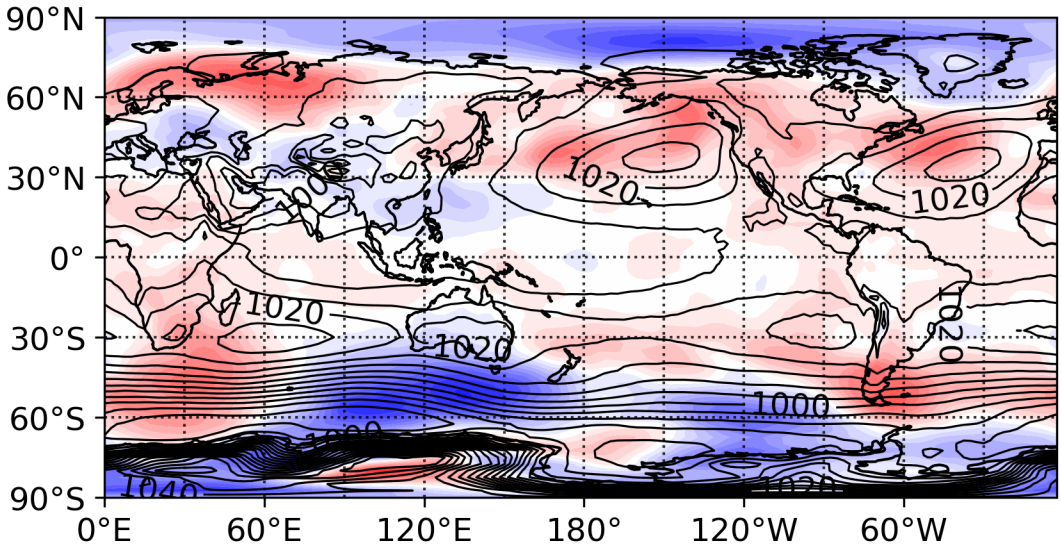


**図 7－2－3　2018 年 7 月の SLP 偏差を陰影で描いた**

### 7.2.4 カラーバーを付ける

作成した図にカラーバーを付け、陰影の値が分かるようにしていきます。作図には cartopy_contourf2.py を用いました。カラーバーを付けるには、5.3.3 節、6.5.3 節で行なったように plt.colorbar を使います。cartopy の場合には、これまでのように plt.contourf、m.contourf の直後に plt.colorbar を行っても、カラーバーの作成に必要な情報がないためエラーになります。代わりに ax.contourf の戻り値を im に格納しておき、plt.colorbar の 1 番目の引数として渡すことでカラーバーを作成します。カラーバーはデフォルトで縦方向ですが、そのままでは図枠の上下に大きくはみ出してしまうので、shrink=0.55 で大きさの調整をしています。カラーバーにラベルを付ける方法は、これまでと同じで plt.colorbar の戻り値 cbar に対して cbar.set_label を使います。

```
im = ax.contourf(x, y, slp_anom, clevs, cmap="bwr")
cbar=plt.colorbar(im, shrink=0.55)
cbar.set_label('SLP anom. in Jul. 2018', fontsize=14)
```

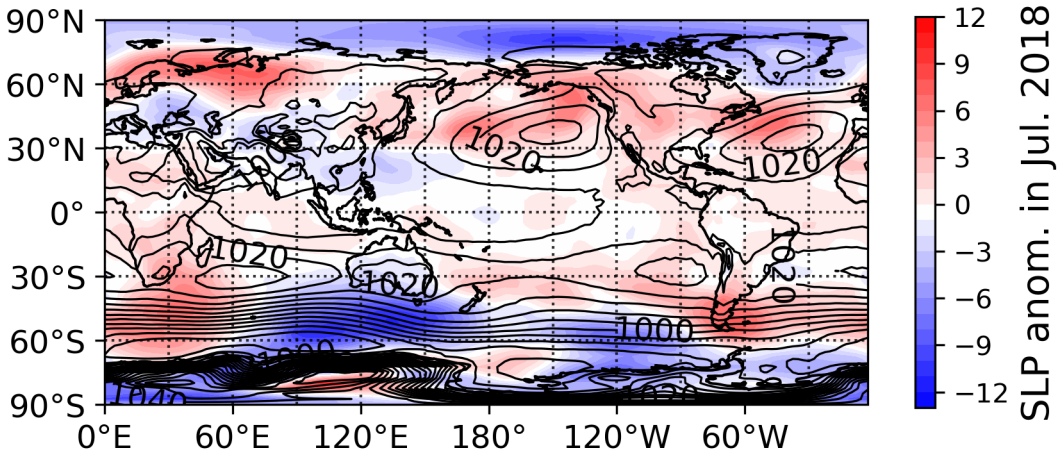


**図７－２－４　カラーバーを付けた**

plt.colorbar では大きさの調整の他にも、表５－３－１に載せた様々なオプションを使用することができます。これらのオプションを使いカラーバーを水平にして大きさやカラーバーとグラフの間の間隔を調整したものが図７－２－５です。作図には cartopy_contourf3.py を用いました。カラーバーは、shrink=0.8 で大きさを調整し、水平方向にするために orientation='horizontal'、カラーバーとグラフの間の間隔を調整するために pad=0.08 を使いました。

```
cbar = plt.colorbar(im, shrink=0.8, orientation='horizontal', pad=0.08)
```

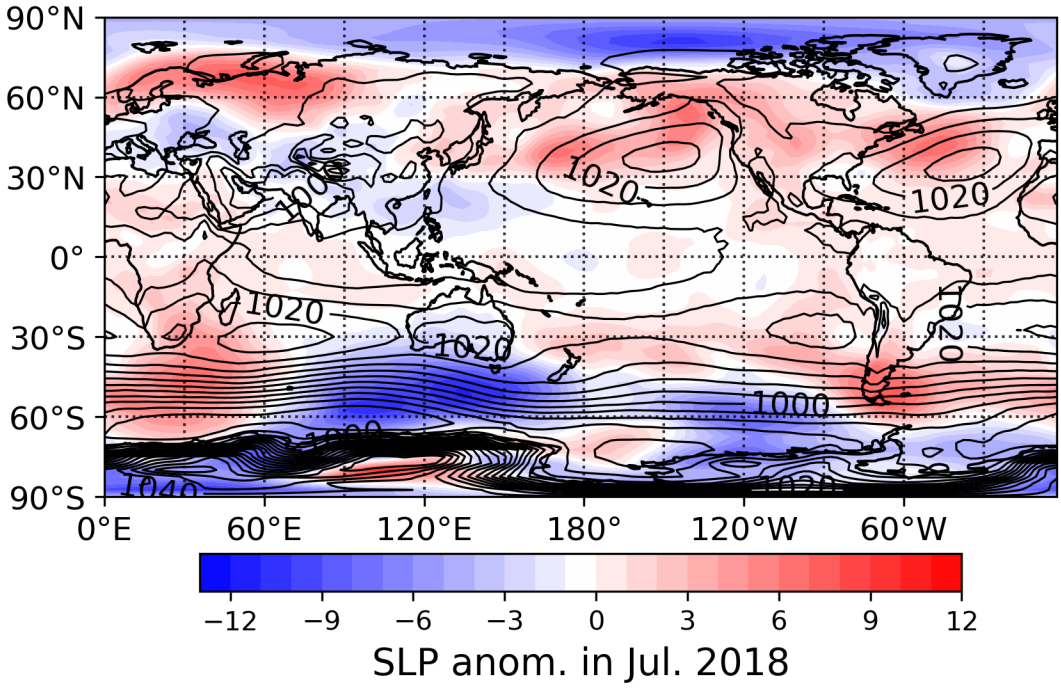


図７－２－５　水平のカラーバーを付けた

cartopy でも matplotlib 同様に、ax.set_title を使い図のタイトルを付けることが可能になっています（図７－２－６）。ここでは fontsize オプションを使い文字を大きくしています。作図には、cartopy_contourf4.py を使いました。

```
ax.set_title("Jul. SLP & anom. in Jul. 2018", fontsize=20)
```

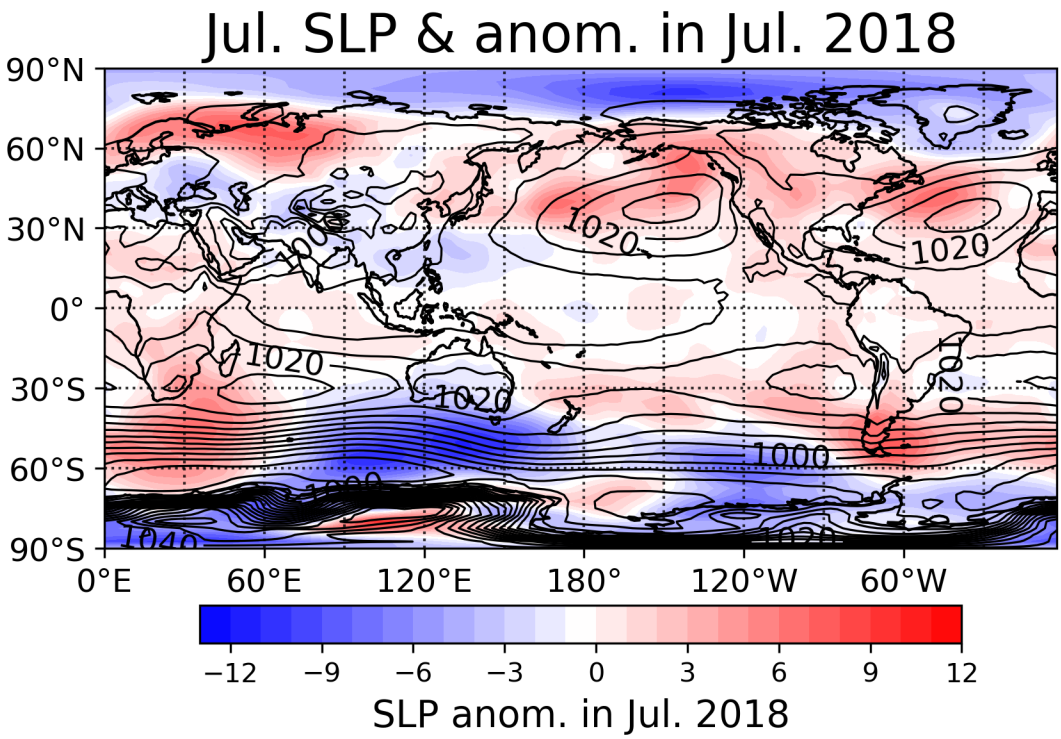


**図７－２－６　タイトルを付けた**

7.2.4 等高線の体裁の調整

図７－２－６では、1000 hPa や 1020 hPa の等高線にラベルが付いていますが、等高線の幅は全て同じになっているため、ラベルのないところでは等高線の値が読み取りにくくなっています。そこで、20 hPa 毎に太線にして等高線の値がわかるように変えてみたいと思います（cartopy_contourf5.py）。等高線の間隔は 4 hPa となっており、5 飛ばしにラベルを付けているため、等高線の幅も 5 飛ばしに太くします。等高線の幅を指定する linewidths オプションでは、linewidths=0.8 のような数値での指定の他に、等高線の幅をリストで与えることも可能です。そこで、linewidths=[1.2, 0.8, 0.8, 0.8, 0.8]のように太線、細線、細線、細線、細線、のリストを与えると、5 飛ばしに太線が描かれるようになります（図７－２－７）。

```
cs = ax.contour(x, y, slp_jul, clevs, linewidths=[1.2, 0.8, 0.8, 0.8, 0.8],
colors='k')
```

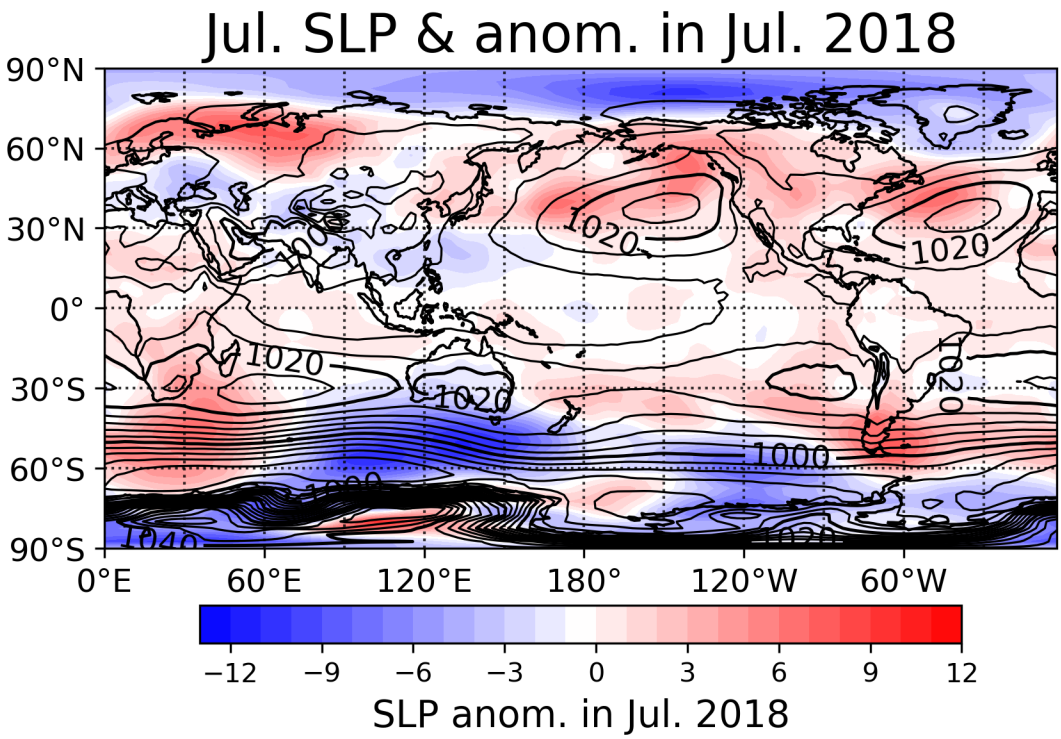


**図７－２－７　20 hPa 毎に太線を描く**

等高線の幅の他に、色（colors）や線種（linestyles）もリストで与えることができ、これらの組み合わせで多様な等高線を描くことが可能です。例えば、linewidths=[1.2, 0.8, 1.2, 0.8]、linestyles=["-", "-", "--", "--"]では、太実線、細実線、太破線、細破線の順で描かれます。

## 7.3 cartopy を使った矢羽、矢印の作図

cartopy では Basemap 同様に矢羽や矢印を描くことができます。ここでは、7.2 節で作成した 7 月の SLP 平均値の等高線に矢羽や矢印を描いていきます。作図には SLP データの NetCDF ファイルに加え、東西風、南北風データの NetCDF ファイル(uwnd.mon.mean.nc、vwnd.mon.mean.nc)も必要なので、ダウンロードしていない場合には basemap_nc2bin.py でダウンロードしておきます。

### 7.3.1 矢羽を描く

矢羽を描くには、ax.barbs(引数 1, 引数 2, 引数 3, 引数 4)を使います。1 番目の引数が x 軸上の位置、2 番目の引数が y 軸上の位置で、3 番目の引数が東西風データ、4 番目の引数が南北風データに当たります。5.1.1 節の plt.barbs と同じなので、表 5－1－1 のオプションが利用可能です。矢羽を描くためのサンプルプログラムは、cartopy_barbs.py です。SLP、東西風、南北風データの 3 ファイルを読み込むために、file_names と var_names にそれぞれのファイル名と NetCDF ファイルに格納されたデータ名を識別するための変数名を定義しておきます。これらを zip に渡してループを回し、flle_name、var_name に同じ番号の要素がセットで入るようにしています。nc.variables で読み込んだデータは、Numpy の ndarray になっており、それらをループが回る毎にリスト d に追加していきます。そのため、リストの要素 d[0]が SLP、d[1]が東西風、d[2]が南北風に対応します。

```
# 読み込むファイル名と変数名
file_names = ["slp.mon.mean.nc", "uwnd.mon.mean.nc", "vwnd.mon.mean.nc"]
var_names = ["slp", "uwnd", "vwnd"]
d = list()
for file_name, var_name in zip(file_names, var_names):
    nc = netCDF4.Dataset(file_name, 'r') # NetCDF データの読み込み
    lon = nc.variables["lon"][:] # 変数の読み込み（経度）
    lat = nc.variables["lat"][:] # 変数の読み込み（緯度）
    d.append(nc.variables[var_name][:]) # 変数の読み込み（データ）
    nc.close() # ファイルを閉じる
```

次に SLP、東西風、南北風それぞれについて、1981～2010 年の 7 月平均値を計算します。例えば SLP の場合、d[0][tstr:tend:12, :, :]のように 1981～2010 年の 7 月のデータを取り出します。最初の d[0]は、リストの 0 番目の要素を取り出すことを意味します。取り出された d[0]は ndarray になっているので、時間軸（axis=0）にスライス記法[tstr:tend:12, :, :]を適用することで、7 月のデータを取り出すことができます。最後の.mean(axis=0)はこれまで同様、時間軸について算術平均を行うことを意味しています。

```
# 1981～2010 年の 7 月平均値
tstr = (1981 - 1948) * 12 + 6
tend = (2010 - 1948 + 1) * 12
slp = d[0][tstr:tend:12, :, :].mean(axis=0) # SLP の 7 月平均値
u = d[1][tstr:tend:12, :, :].mean(axis=0) #  東西風の 7 月平均値
v = d[2][tstr:tend:12, :, :].mean(axis=0) #  南北風の 7 月平均値
```

7.2.1 節、7.2.2 節と同様に等高線を描いてラベルを付けた後、ax.barbs に color='r'オプションを与えることで赤色の矢羽を描いています（図 7－3－1）。また、6.5.5 節の Basemap の所で解説した矢羽を間引く方法（3 グリッド毎に描くために[::3, ::3]の記法を使う）、風速の小さい場所に丸印が描かれないようにする方法（sizes=dict(emptybarb=0.0)オプション）も使いました。なお、矢羽を描くために用いる東西風、南北風データの単位が m/s なので、短矢羽が 5 m/s、長矢羽が 10 m/s、旗矢羽が 50 m/s で描かれます。

```
ax.barbs(x[::3, ::3], y[::3, ::3], u[::3, ::3], v[::3, ::3],
            sizes=dict(emptybarb=0.0), length=4, color='r')
```

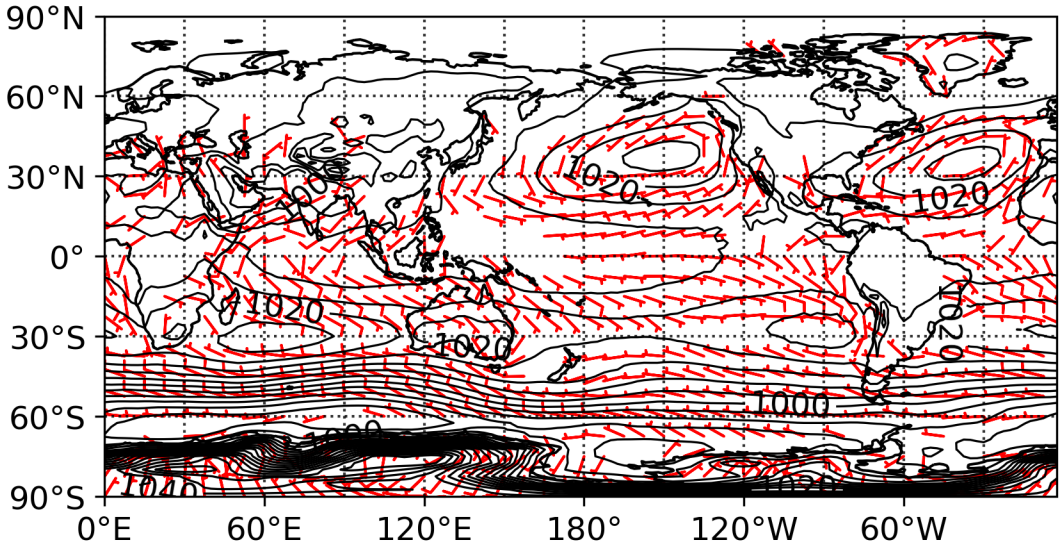


図７－３－１　1981～2010 年の７月平均値から矢羽を描く

7.3.2 矢印を描く

矢印を描くには、ax.quiver(**引数１**, **引数２**, **引数３**, **引数４**)を使います。引数は矢羽の場合と同じで、x 軸上の位置、y 軸上の位置、東西風データ、南北風データになります。矢印を描くプログラムは cartopy_quiver.py です。ax.quiver は 5.1.4 節の plt.quiver と同じなので、表５－１－２のオプションが利用可能です。ここでは、色を指定する color オプションを付けて赤矢印を描きました（図７－３－２）。

```
ax.quiver(x[::3, ::3], y[::3, ::3], u[::3, ::3], v[::3, ::3], color='r')
```

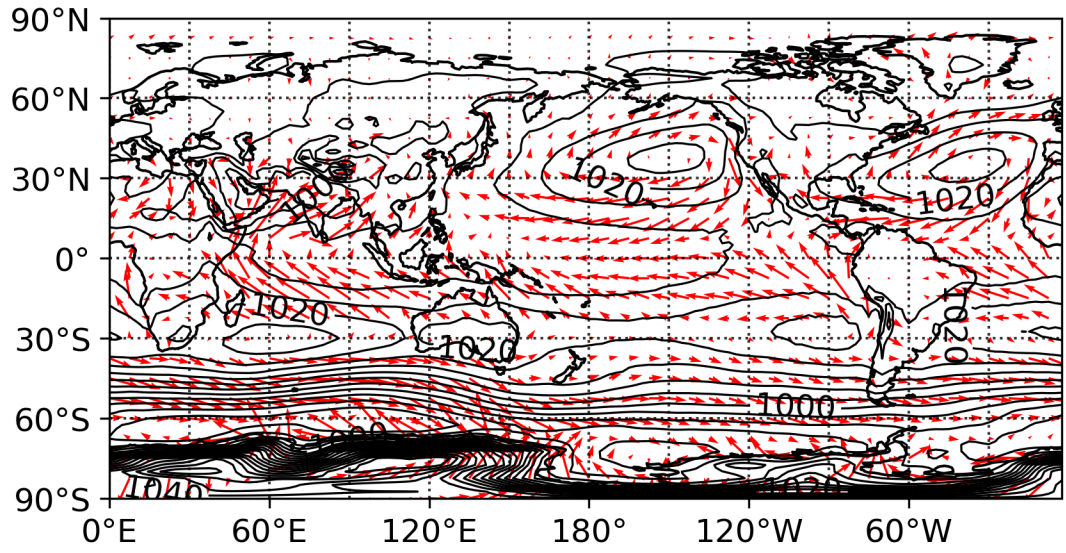


**図７－３－２　1981～2010 年の７月平均値から矢印を描く**

これだけでは矢印の大きさが分からないので、凡例を付けてみます。まずは、矢印の幅と長さを固定するため、width=.003、scale=300.を指定しました。作図には cartopy_quive2.py を使いました。

```
Q = ax.quiver(x[::3, ::3], y[::3, ::3], u[::3, ::3], v[::3, ::3], ¥
    color='r', width=.003, scale=300.)
```

矢印の凡例を描くには、ax.quiverkey を用います。１番目の引数の Q は ax.quiver の戻り値です。２番目と３番目の引数は、図枠の経度・緯度範囲のど

の場所に凡例を描くのかを表します。図枠の範囲が 0 ～ 1 に対応し、左下が(0.0, 0.0)、右上が(1.0, 1.0)です。４番目の引数の整数 10 は、10m/s の矢印を描くことを表し、５番目の引数の文字列'10'は、矢印に表示する文字列を表します。labelpos は、矢印に対してどの位置に文字列を描くのかを意味し、labelpos='N'で矢印の上側、labelpos='S'で下側に描かれます（デフォルトでは labelpos='N'）。矢印と文字列との間隔は labelsep で指定します（デフォルトは labelsep =0.1）。デフォルトでは矢印と文字列が離れすぎているので、0.01 と小さな値にしました。図７－３－３では、図枠の右上に横向きの矢印とその上に 10 の文字が描かれています。

```
ax.quiverkey(Q, 0.97, 0.93, 10, '10', labelpos='N', color='r', labelsep=0.01)
```

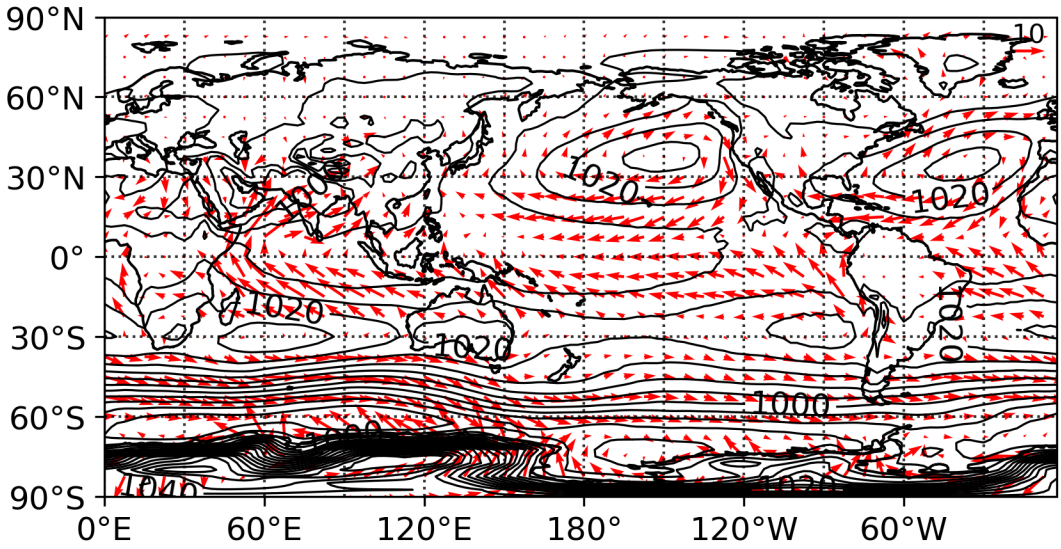


**図７－３－３　矢印の凡例を付ける**

この状態では凡例の判別が難しいので、凡例の近くを塗り潰してみます（図７－３－４）。凡例の周りが灰色の背景になったのが分かるでしょうか。ここまでは矢印を赤で描いていましたが、黒に変えました。また、矢印の頭の部分の幅を大きくして方向が判別しやすいように変えました（headwidth=3.75）。図の上に来るようにするため zorder=4 としています。作図には cartopy_quive3.py を使いました。

```
Q = ax.quiver(x[::3, ::3], y[::3, ::3], u[::3, ::3], v[::3, ::3], ¥
      color='k', width=.003, scale=300., headwidth=3.75, zorder=4)
```

凡例の周りの背景を灰色に塗り潰すため、ax.add_patch で矩形領域を追加しました。矩形領域の設定は、plt.Rectangle で行いました。最初の(155, 70)は東経 155 度、緯度 70 度を左下の始点とすることを意味し、次の 25 と 20 は、経度方向に 25 度、緯度方向に 20 度の幅を持つことを意味します。そのような設定なので、東経 155～180 度、緯度 70～90 度が塗り潰されることになります。塗り潰す色は facecolor で指定し、ここでは灰色（facecolor='gray'）としました。塗り潰した領域の端には枠線を付けないようにするため、edgecolor=None としています。

```
rect = plt.Rectangle((155, 70), 25, 20, facecolor='gray', ¥
                              edgecolor=None, zorder=4)
ax.add_patch(rect)
```

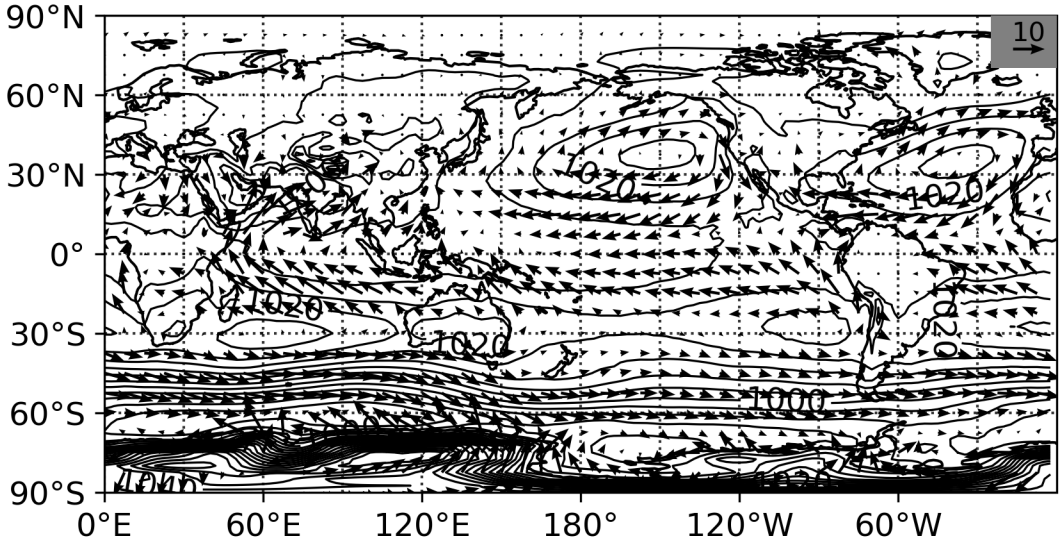


**図７－３－４　左上を灰色で塗り潰し凡例を付ける**

図枠の内側が塗り潰されると困る場合もあると思います。そのような場合には、矢印の凡例を図枠の外に出すことも可能です（図７－３－５）。作図には

cartopy_quive4.py を使いました。凡例の緯度方向の位置を表す３番目の引数にマイナスの値を与えると、図枠の下側に表示されます。矢印のラベルを矢印の下側に表示するため、labelpos='S'としました。

```
ax.quiverkey(Q, 0.97, -0.04, 10, '10', labelpos='S', color='k', labelsep=0.03)
```

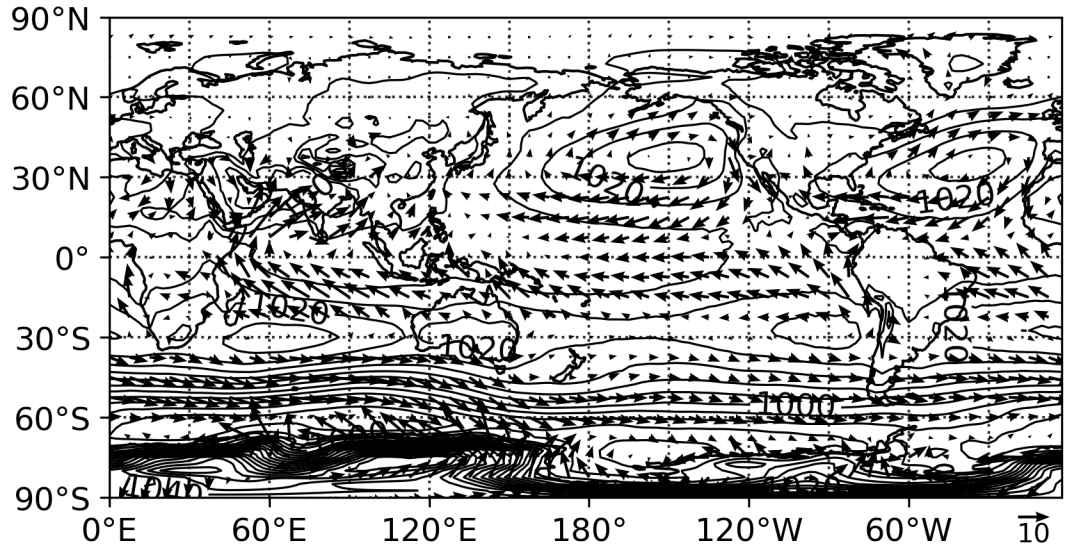


**図７－３－５　図枠の左下に凡例を付ける**

ラベルの文字の大きさを変えることも可能です（図７－３－６）。ここでは判例に用いる風速を 15m/s に変え、文字の大きさを 12 にしました。設定は、fontproperties={'size': 12}です。作図は cartopy_quive5.py で行いました。なお、coordinates='axes'はデフォルト値で、省略しても同じですが、x 軸上の位置、y 軸上の位置を図枠の範囲で０～１として指定することを意味します。

```
ax.quiverkey(Q, 0.97, -0.04, 15, '15', labelpos='S', color='k', labelsep=0.03,
                coordinates='axes', fontproperties={'size': 12})
```

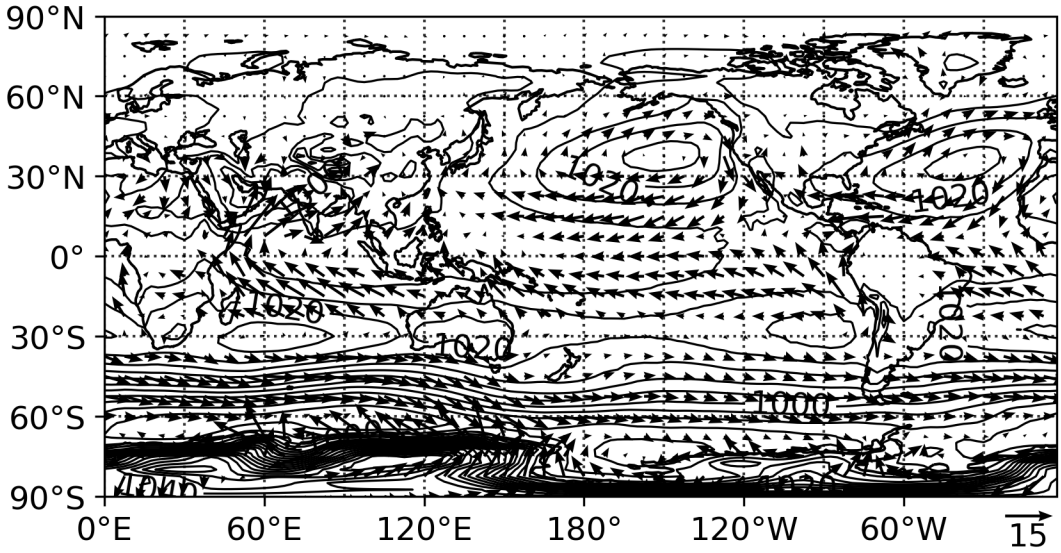


図７－３－６　ラベルの文字を大きくする

## 7.4 cartopy で利用できる図法

cartopy でも Basemap のように、様々な図法を利用して作図することができるようになっています。ここでは cartopy で作図する方法についてのみ解説するので、図法については 6.2 節の解説を参照してください。

### 7.4.1 正射投影図法

6.2.2 節で紹介した正射投影図法（あるいは平射図法）で作図する方法です。作図を行うプログラムは、cartopy_proj.py です。正射投影図法で作図するには、projection オプションに ccrs.Orthographic()を与えます。図 7－4－1 のように地球儀風の図ができました。Basemap の所で行った図 6－2－1 の右上の図の場合と同様に、東経 180 度、北緯 45 度を中心に描いたので、北太平洋が手前側に来ています。

```
ax = fig.add_subplot(1, 1, 1, ¥
                projection=ccrs.Orthographic(central_longitude=180.0, ¥
                central_latitude=45.0))
ax.set_title("projection='ccrs.Orthographic'") #  タイトルを付ける
```

正射投影図法の場合にも経度線・緯度線を描く間隔を指定でき、次のように経度線・緯度線を 10 度毎に付けるようにしました。ラベルを付けることはできないため、ラベルの設定はありません。

```
#  緯度線、経度線を描く値
dlon, dlat = 10, 10
xticks = np.arange(0, 360.1, dlon)
yticks = np.arange(-90, 90.1, dlat)
gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=False, ¥
                linewidth=1, linestyle=':', color='k', alpha=0.8)
gl.xlocator = mticker.FixedLocator(xticks) #  経度線を描く値
gl.ylocator = mticker.FixedLocator(yticks) #  緯度線を描く値
```

さらに、ax.set_title で図にタイトルを付けました。

```
ax.set_title("projection='ccrs.Orthographic'") #  タイトルを付ける
```

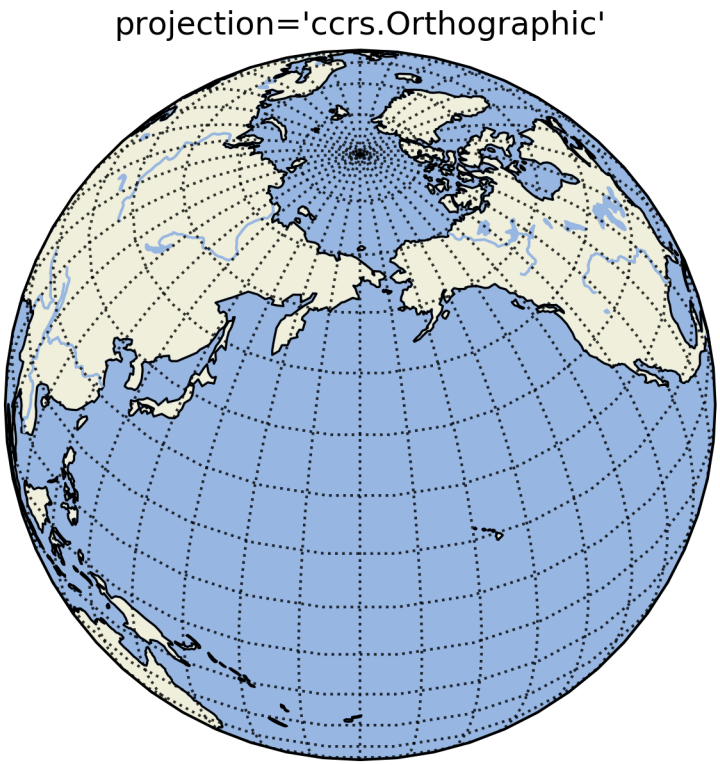


図７－４－１　正射投影図法で描く

7.4.2 メルカトル図法

6.2.3 節で紹介したメルカトル図法でも作図することができます（図７－４－２）。作図を行うプログラムは、cartopy_proj2.py です。メルカトル図法で作図するには、projection オプションに ccrs.Mercator()を与えます。ここでは、太平洋側を中心にするため中心経度 central_longitude=180.0 とし、南緯 60 度から北緯 60 度の範囲で描くため、min_latitude=-60.0、max_latitude=60.0 としました。経度線は 30 度毎、経度線は 15 度毎に描きました。

```
ax = fig.add_subplot(1, 1, 1, ¥
                     projection=ccrs.Mercator(central_longitude=180.0, ¥
                                              min_latitude=-60.0, ¥
                                              max_latitude=60.0))
ax.set_title("projection='ccrs.Mercator'")
```

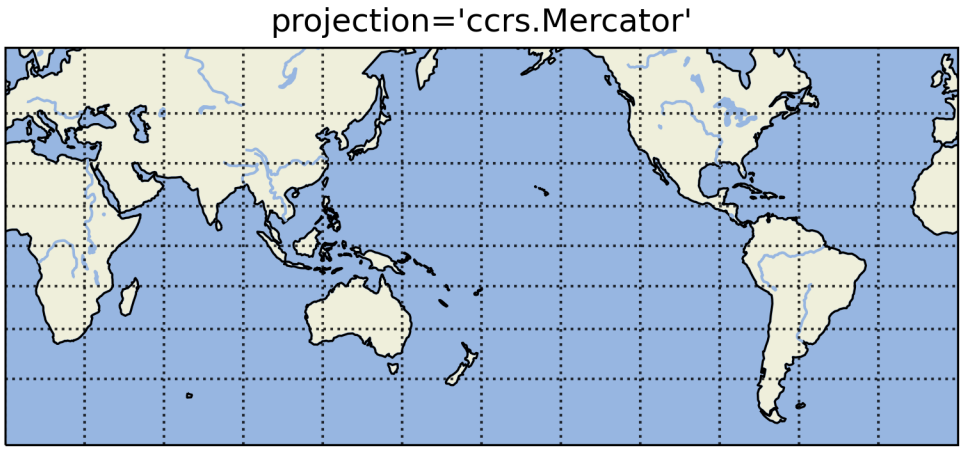


**図７－４－２　メルカトル図法で描く**

メルカトル図法の作図では、東経 0 度が中心の図では、ax.gridlines で draw_labels=True とすれば、目盛り線ラベルを付けることができます。中央を東経 180 度とした場合には、経度帯の半分しか目盛り線ラベルがつかないため、ここでは、ラベルを付けていません。

### 7.4.3 ランベルト正角円錐図法

6.2.4 節で紹介したランベルト正角円錐図法も試してみます（図７－４－３）。作図を行うプログラムは、cartopy_proj3.py です。ランベルト正角円錐図法で作図するには、projection オプションに ccrs.LambertConformal()を与えます。ランベルト正角円錐図法では、図の中心経度と中心緯度の設定によって図の歪みなどが変わってくるため、対象となる領域に近い点を指定した方が良いでしょう。ここでは、6.2.4 節の場合と同じ、中心経度 central_longitude=180.0、中心緯度 central_latitude=60.0 を使いました。

```
ax = fig.add_subplot(1, 1, 1, ¥
                projection= ccrs.LambertConformal(central_longitude=180.0, ¥
                                                  central_latitude=50.0))
ax.set_title("projection='ccrs.Mercator'")
```

ランベルト正角円錐図法の場合には、ax.gridlines で draw_labels=True とした場合、バージョン 0.19 では緯度・経度のラベルが表示されますが、古いバージョンではエラーとなります。

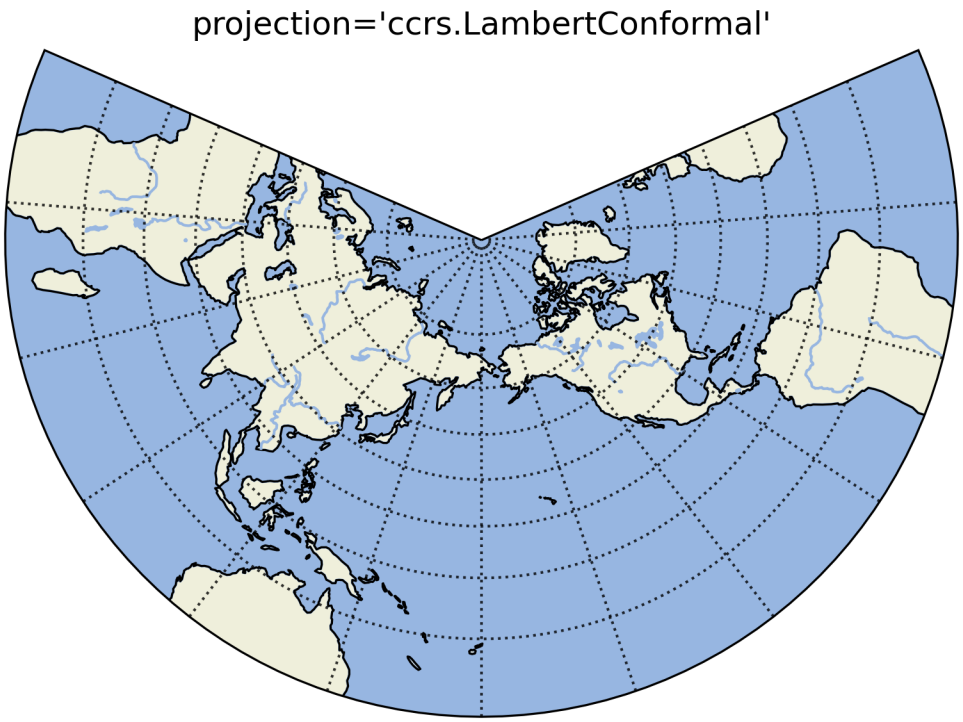


**図７－４－３　ランベルト正角円錐図法で描く**

ランベルト正角円錐図法では両極を同時に描けないので、地球全体を把握するための図よりは領域を限定した図に適しています。ax.set_extent を使うことで日本周辺に限定することもできます（図７－４－４）。作図は cartopy_proj4.py で行いました。中心経度は東経 135 度、中心緯度は北緯 35 度に設定しました。dlon、dlat を変更し、経度線、経度線は 10 度毎に描きました。

```
ax = fig.add_subplot(1, 1, 1, ¥
                projection=ccrs.LambertConformal(central_longitude=135.0,¥
                                                 central_latitude=35.0))
ax.set_extent([100, 170, 10, 70]) #  領域の限定
```

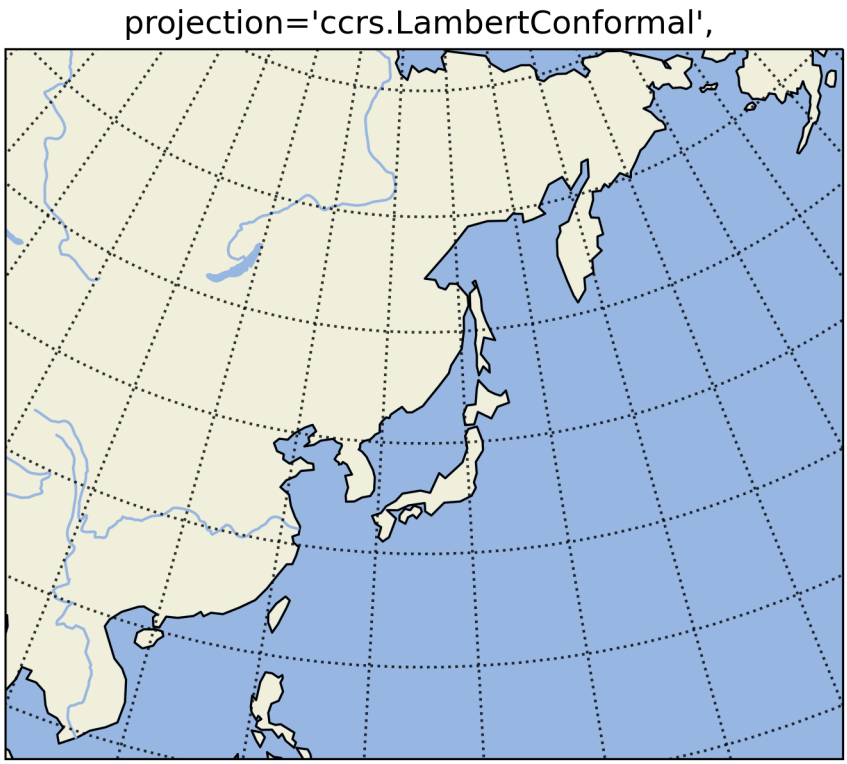


**図７－４－４　ランベルト正角円錐図法で日本周辺だけを描く**

### 7.4.4 極投影図法

6.2.5 節で紹介した極投影図法（ポーラーステレオ図法）で描くことも可能です（図７－４－５）。作図は cartopy_proj5.py で行いました。南極を中心とする極投影図法で作図する場合、projection オプションに ccrs.SouthPolarStereo() を与えます。北極を中心とする場合には、ccrs.NorthPolarStereo()です。極投影図法では作図する領域を指定する必要があり、ここでは、南極から南緯 20 度までの全経度帯を ax.set_extent([-180, 180, -90, -20], ccrs.PlateCarree())で指定しています。ここで ccrs.PlateCarree()を省略するとうまく作図できません。dlon, dlat = 30, 10 に設定して経度線を 30 度毎、緯度線を 10 度毎に描きました。指定したのは南緯 20 度までですが、図７－４－５のように、南緯 20 度の緯度円に外接するような四角形の領域まで描かれます。

```
ax = fig.add_subplot(1, 1, 1, projection=ccrs.SouthPolarStereo())
ax.set_extent([-180, 180, -90, -20], ccrs.PlateCarree())
ax.set_title("projection='ccrs.SouthPolarStereo'")
```

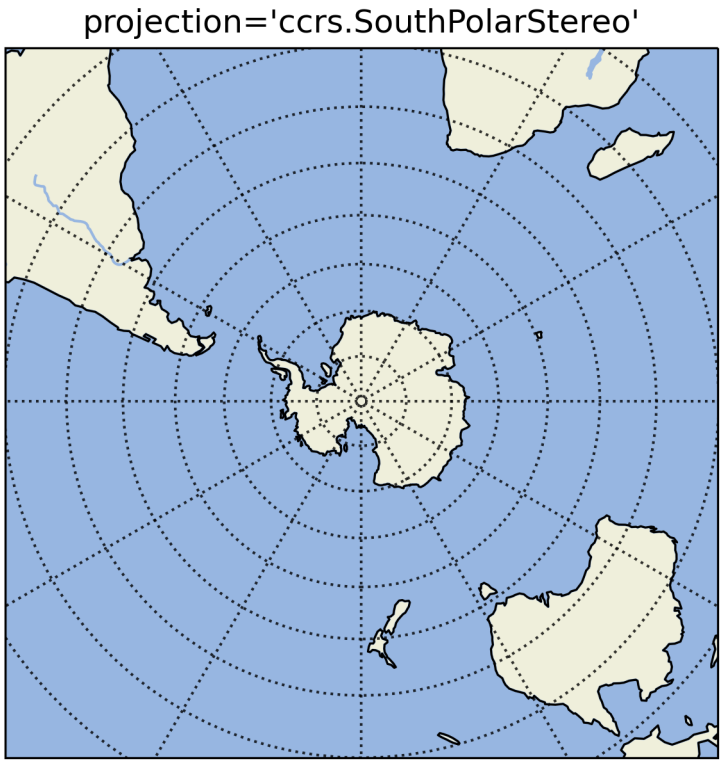


**図７－４－５　極投影図法で南極周辺を描く**

cartopy では、作図範囲を南緯 20 度の緯度円までに限定することができ、cartopy_proj6.py で行っています（図 7－4－6）。cartopy を呼び出した際のサブプロット（軸：axes）は、左下の座標が（0.0, 0.0)、右上の座標が（1.0, 1.0）です（図 7－4－7）。サブプロットの端が図 7－4－5 の図枠に対応しており、この領域は ax.transAxes で参照することができます。そのため、中心座標を（0.5, 0.5）とした半径 0.5 の円を作成すれば、この円は図枠の四角形と接することが分かります。円形の領域内だけ表示させることができれば、南緯 20 度の緯度円に限定できるようになります。ここでは、matplotlib に含まれている matplotlib.path を使い、このモジュールを mpath で参照しています。mpath.Path に円形の領域を渡して領域内だけ表示させることで、南緯 20 度の緯度円に限定しています。まず center に中心（0.5, 0.5）の座標を、radius に半径 0.5 を入れています。次に 0 ～2π の範囲で位相 theta を作成し、中心 (0.0, 0.0)、半径 1 の円を構成する座標を計算して、verts に格納しています。最後に、verts * radius + center の計算を行った結果を mpath.Path に渡し、円形の領域を作成します（circle)。ax.set_boundary は、図枠の領域を限定するために使っており、範囲としてサブプロットの四角形の領域に内接する円形の領域 circle を渡すことで、南緯 20 度の緯度円になります。

```
import matplotlib.path as mpath
center, radius = [0.5, 0.5], 0.5
theta = np.linspace(0, 2 * np.pi, 100)
verts = np.vstack([np.sin(theta), np.cos(theta)]).T
circle = mpath.Path(verts * radius + center)
ax.set_boundary(circle, transform=ax.transAxes)
```

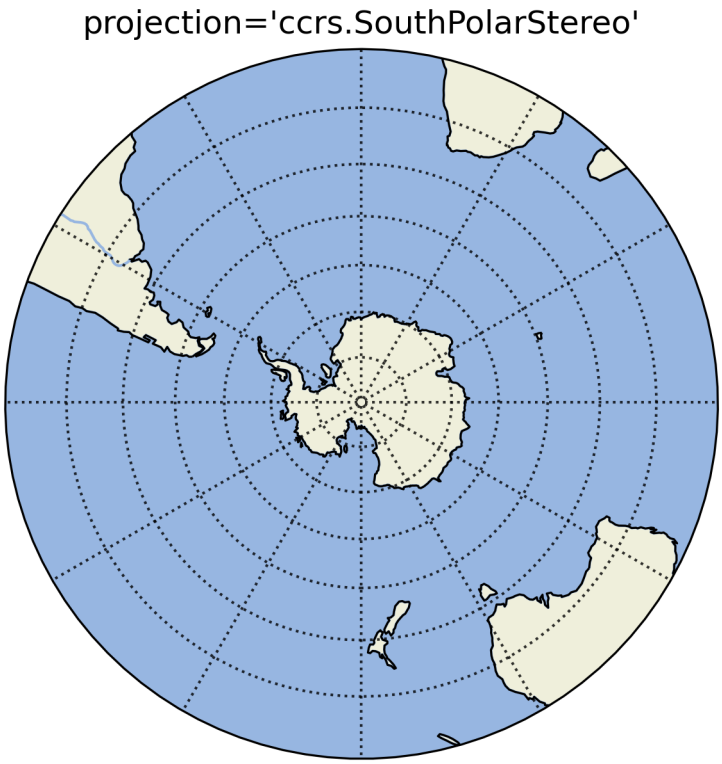


図７－４－６　作図範囲を南緯 20 度の緯度円までに限定する

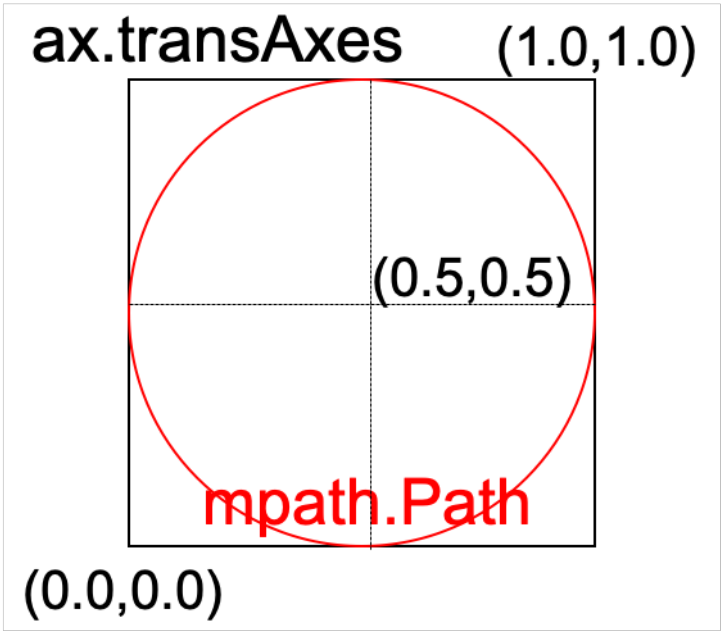


図７－４－７　ax.transAxes の領域と mpath.Path で設定した円形領域との関係

7.4.5 モルワイデ図法

6.2.6 節で紹介したモルワイデ図法も試すことができます（図７－４－８）。作図には、cartopy_proj7.py を使いました。モルワイデ図法の場合、projection オプションに ccrs.Mollweide()を与えます。太平洋側を中心にするために、中心経度 central_longitude=180.0 としました。dlon, dlat = 30, 30 と設定し、経度線、経度線を 30 度毎に描きました。

```
ax = fig.add_subplot(1, 1, 1, ¥
                     projection=ccrs.Mollweide(central_longitude=180.0))
ax.set_title("projection='ccrs.Mollweide'")
```

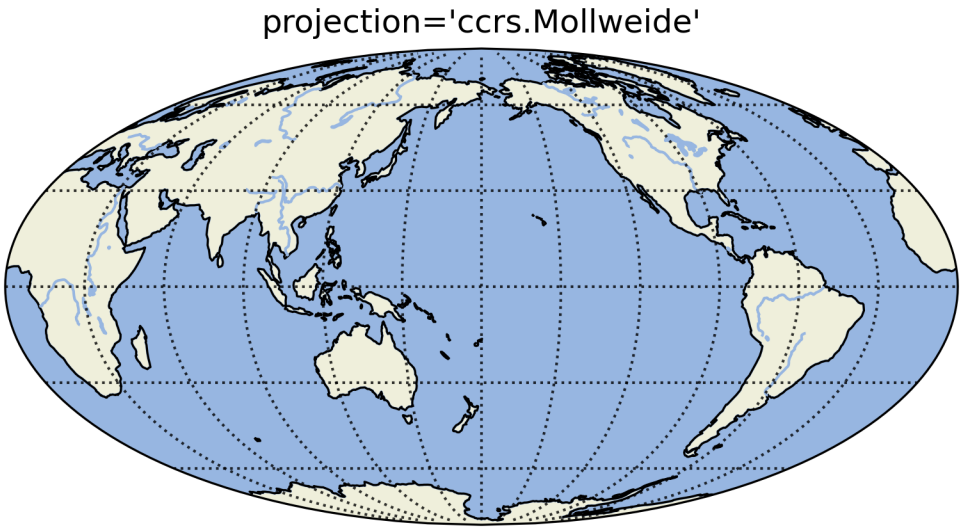


**図７－４－８　モルワイデ図法で描く**

### 7.4.6 ロビンソン図法

6.2.7 節で紹介したロビンソン図法も試すことができます（図７－４－９）。ロビンソン図法の場合には、projection オプションに ccrs.Robinson()を与えます。cartopy_proj8.py で作図しました。

```
ax = fig.add_subplot(1, 1, 1, ¥
                     projection=ccrs.Robinson(central_longitude=180.0))
ax.set_title("projection='ccrs.Robinson'")
```

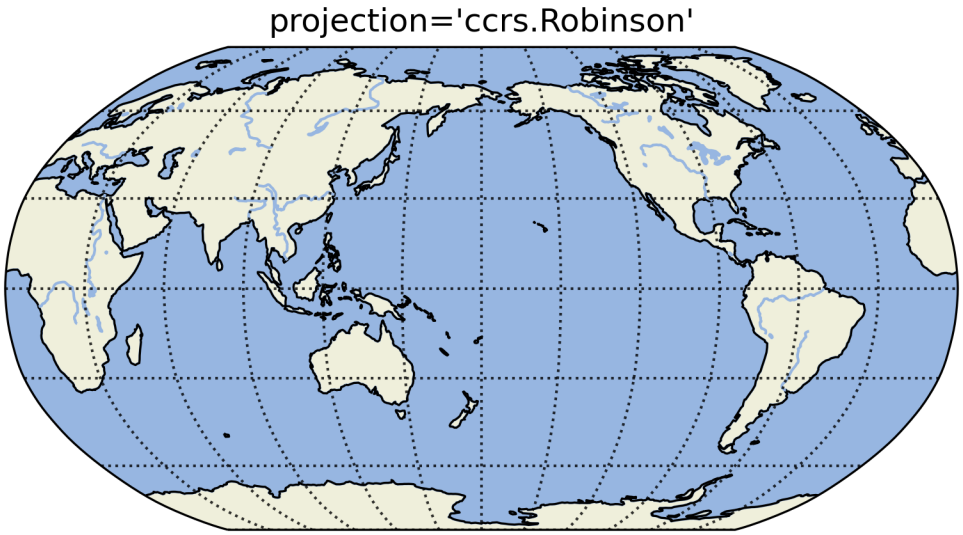


**図７－４－９　ロビンソン図法で描く**

### 7.4.7 ランベルト正積円筒図法

6.2.8 節で紹介したランベルト正積円筒図法も利用できます（図７－４－10）。作図には、cartopy_proj9.py を用いました。ランベルト正積円筒図法の場合には、projection オプションに ccrs.LambertCylindrical()を与えます。dlon, dlat = 30, 15 と設定し、経度線を 30 度毎に、経度線は極域ほど幅が狭くなっていることが分かるように 15 度毎に描きました。

```
ax = fig.add_subplot(1, 1, 1, ¥
                projection=ccrs.LambertCylindrical(central_longitude=180.0))
ax.set_title("projection='ccrs.LambertCylindrical'")
```

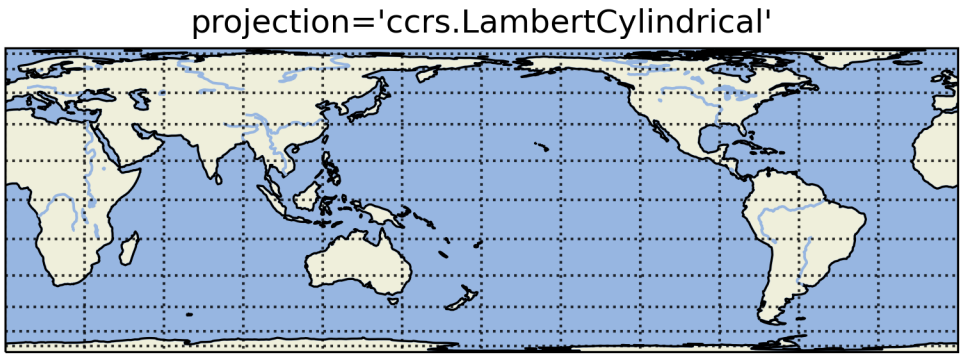


**図７－４－10　ランベルト正積円筒図法で描く**

### 7.4.8 ミラー図法

6.2.9 節で紹介したミラー図法で描く機能もあります（図７－４－11）。ミラー図法で描くには、projection オプションに ccrs.Miller()を与えます。赤道付近は図７－４－２と似ていますが、両極まで描くことが可能になります。作図には、cartopy_proj10.py を用いました。

```
ax = fig.add_subplot(1, 1, 1,
                     projection=ccrs.Miller(central_longitude=180.0))
ax.set_title("projection='ccrs.Miller'")
```

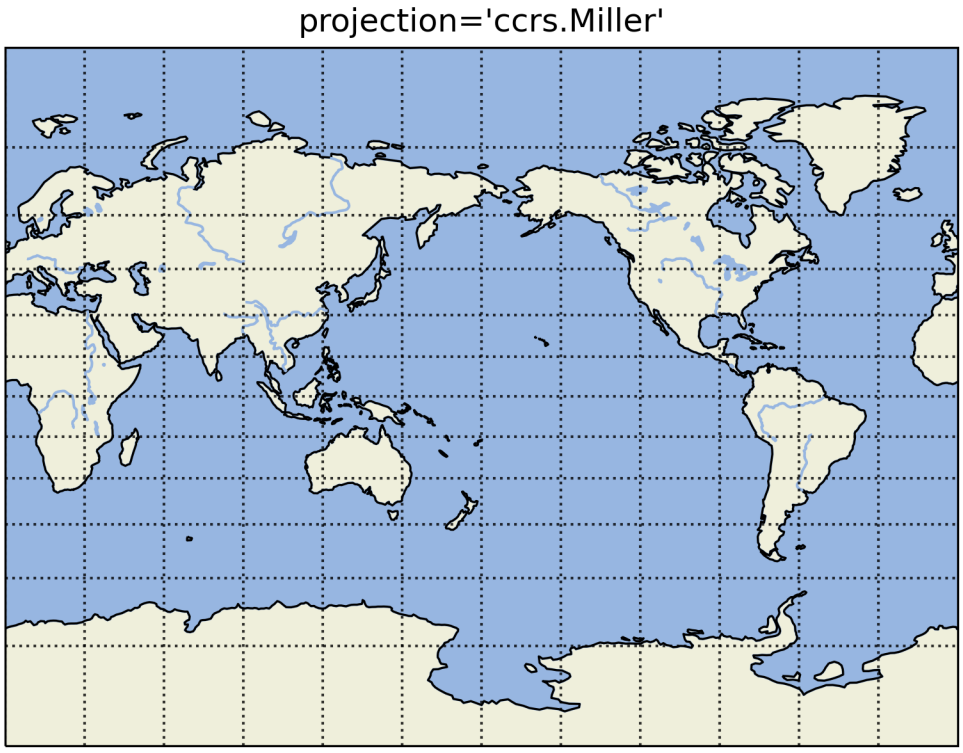


**図７－４－11　ミラー図法で描く**

## 7.5 cartopy バージョン 0.17

バージョン 0.17 から 0.18 への更新で経度線や緯度線等の作図が変更され、互換性のあるコードを作成することが難しくなりました。バージョン 0.18 以降の便利な機能を利用するのが望ましいですが、以前のバージョンしか利用できない環境のことも考え、本節ではバージョン 0.17 で作図を行う場合のポイントについてまとめています。

### 7.5.1 経度線や緯度線作図時の挙動

7.1.2 節で行った経度線と緯度線の作図は、バージョン 0.17 でも ax.gridlines で行い、経度線や緯度線を引く値は、gl.xlocator と gl.ylocator で与えます。

```
import matplotlib.ticker as mticker
gl = ax.gridlines(crs=ccrs.PlateCarree())
gl.xlocator = mticker.FixedLocator(np.arange(0, 360, 30)) #  経度線
gl.ylocator = mticker.FixedLocator(np.arange(-90, 90, 30)) #  緯度線
```

バージョン 0.17 の場合、cartopy_0.17_sample.py で作図すると、図 7－5－1 のようになります。右端付近（東経 360 度付近）と上端付近（北極付近）の経度線・緯度線が描かれていないのが分かると思います。

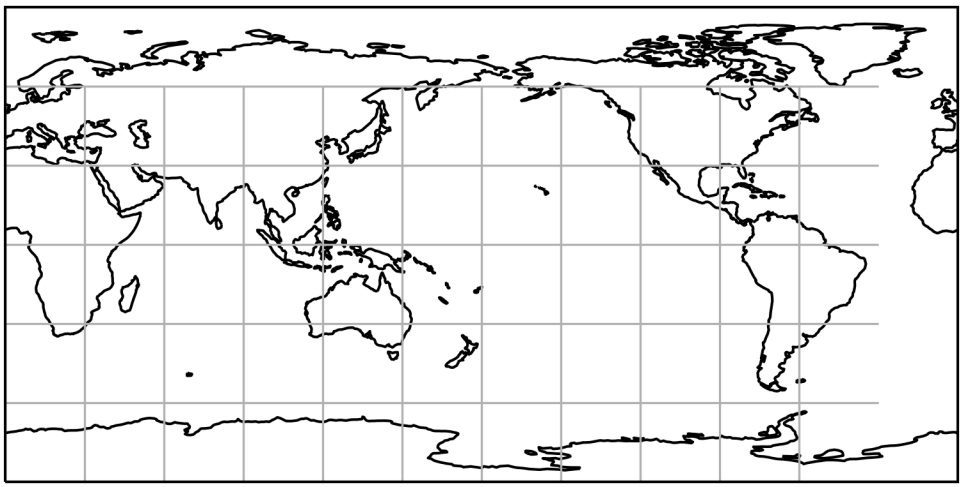

**図 7－5－1　cartopy0.17 で経度線・緯度線を描く**

経度線を引く値 np.arange(0, 360, 30)は、開始点の 0 から終了点の 360 まで 30 毎の値を並べることを意味していますが、終了点の値は出力されません（0., 30., …, 330.のように最後の値が 330）。バージョン 0.17 の場合、このような値を渡すと東経 330 度までしか描かれません。東経 360 度まで出力するには、np.arange(0, 360.1, 30)とする必要があります。

cartopy_0.17_sample2.py では、経度線を描く値を 360.1 まで、緯度線を描く値を 90.1 までに修正しています。バージョン 0.17 で動かすと、図 7－5－2 のように、右端付近や上端付近でも経度線・緯度線が引かれるようになります。

```
gl = ax.gridlines(crs=ccrs.PlateCarree())
gl.xlocator = mticker.FixedLocator(np.arange(0, 360.1, 30)) #  経度線
gl.ylocator = mticker.FixedLocator(np.arange(-90, 90.1, 30)) #  緯度線
```

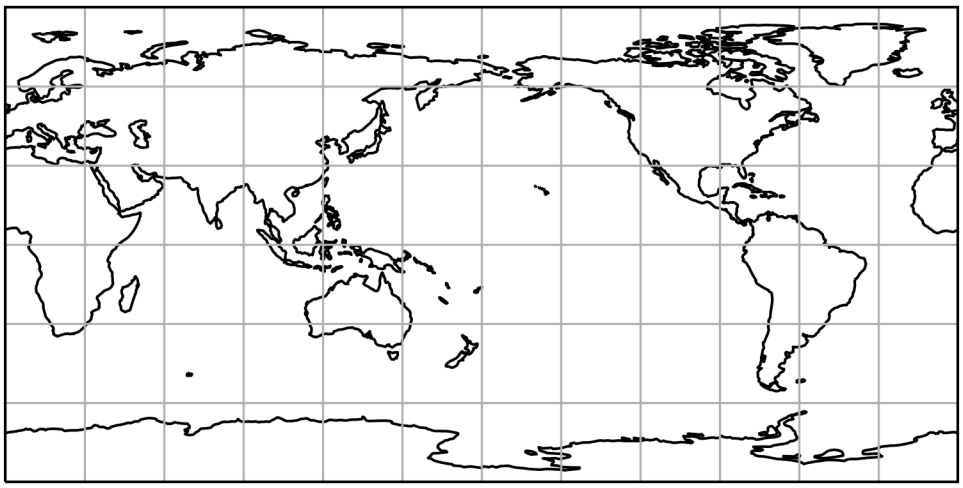

**図 7－5－2　バージョン 0.17 で全ての経度線・緯度線が描かれるように修正**

7.5.2 経度線・緯度線ラベル作図時の挙動

7.1.4 節で紹介した経度線・緯度線ラベルの作図では、draw_labels オプションを使いました。cartopy_0.17_sample3.py では、経度線の範囲を np.arange(-180, 180.1, 30)で西経 180 度から東経 180 度まで 30 度毎に設定しました。バージョン 0.17 で作図したものが、図７－５－３です。

```
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree()) # cartopy 呼び出し
gl.xlocator = mticker.FixedLocator(np.arange(-180, 180.1, 30)) #  経度線
gl.ylocator = mticker.FixedLocator(np.arange(-90, 90.1, 30)) #  緯度線
```

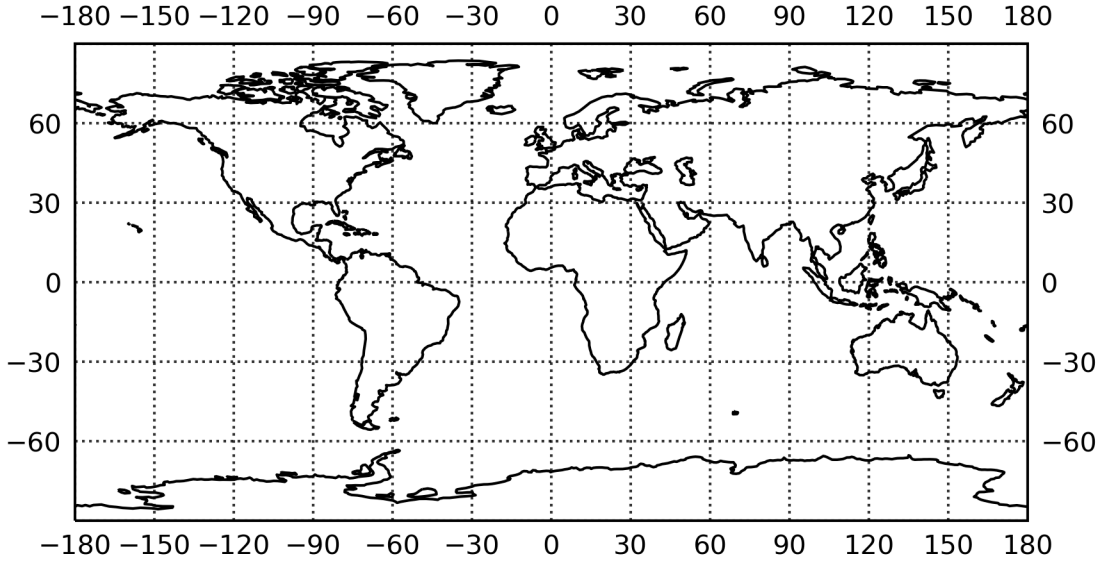


**図７－５－３　バージョン 0.17 で経度線・緯度線にラベルを付ける**

東経０度を中心とした場合にはうまくラベルが付くものの、他の範囲では失敗します。試しに東経 180 度を中心とした場合には、図７－５－４のように 180 度より東側ではラベルが付かないという不具合が起こります。

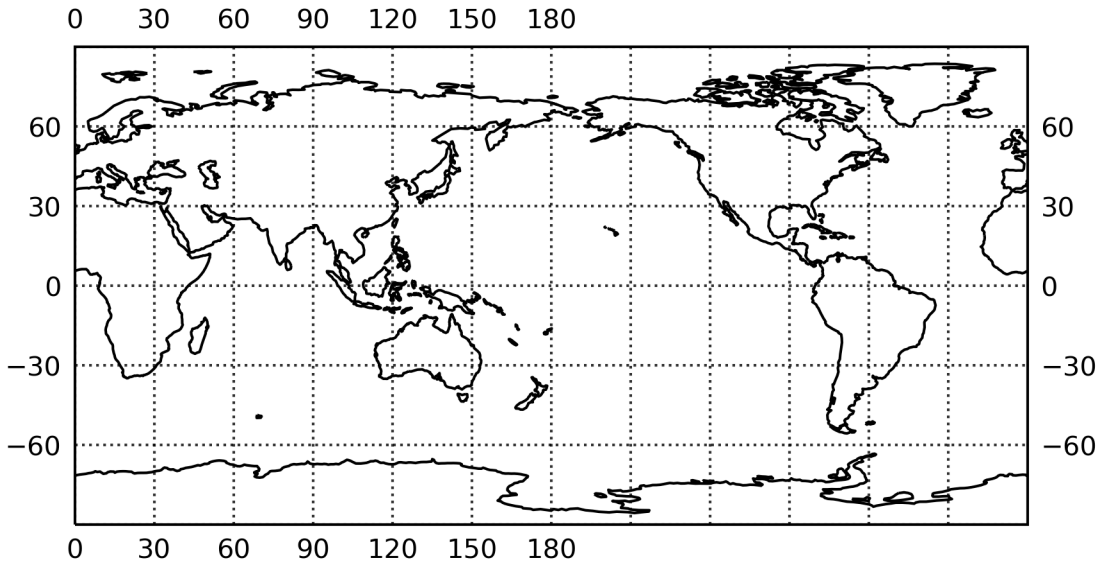


**図７－５－４　バージョン 0.17 で東経 180 度を中心とした場合**

ラベルを描く際に、7.1.5 節、7.1.6 節で紹介した ax.set_xticks、ax.set_yticks を使えば、バージョン 0.17 でもうまくラベルをつけることができます。ここでは、経度線を 30 度毎、経度線ラベルを 60 度毎に描いた図７－１－11 を 0.17 で描いてみます（図７－５－５）。作図に用いた cartopy_0.17_sample4.py では、np.arange に与える経度範囲を(0, 360.1)に変える必要があります。なお 0.18 以降では、cartopy_sample11.py のように(-180, 180)の範囲で与えていました。

```
# 緯度線、経度線とラベルを描く値
dlon, dlat = 30, 30
xticks = np.arange(0, 360.1, dlon)
yticks = np.arange(-90, 90.1, dlat)
```

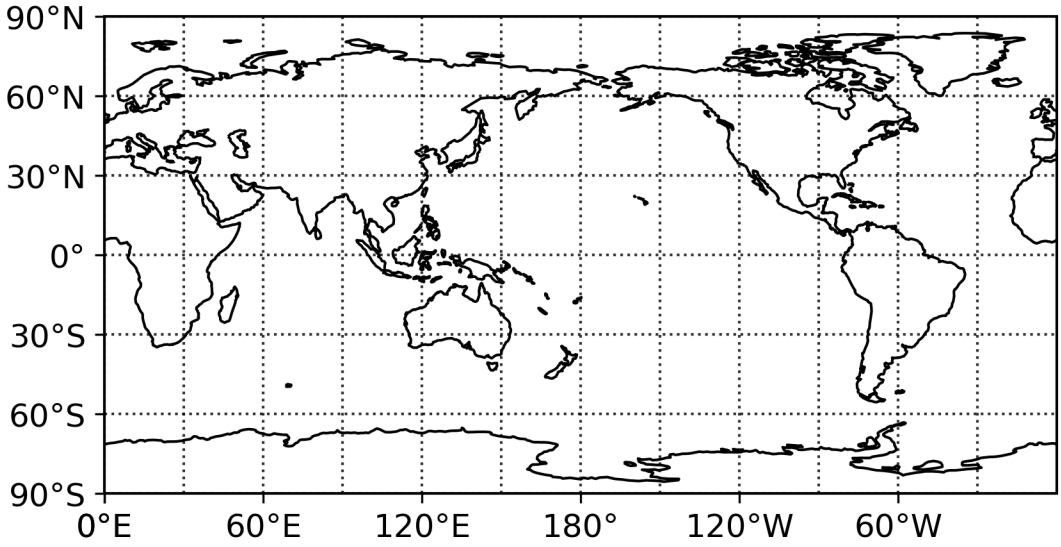


図７－５－５　バージョン 0.17 で経度線を 30 度毎、経度線ラベルを 60 度毎に描く