Add Folders to Path

In [1]:
```
%%time
import sys, os
# get current directory
path = os.getcwd()
# get parent directory
parent_directory = os.path.sep.join(path.split(os.path.sep)[:-2])
# add Algorithm folder to current working path in order to access the functions inside the folder ´Algorithms´
sys.path.append(parent_directory+"/General_Functions")
```

```
CPU times: user 129 µs, sys: 97 µs, total: 226 µs
Wall time: 158 µs
```

Agulhas

Import data

In [2]:
```
%%time
import scipy.io as sio

#Import velocity data from file in data-folder
mat_file = sio.loadmat('../../Data/Agulhas_AVISO.mat')

U = mat_file['u']
V = mat_file['v']
x = mat_file['x']
y = mat_file['y']
time = mat_file['t']
```

```
CPU times: user 143 ms, sys: 38.6 ms, total: 182 ms
Wall time: 182 ms
```

Data/Parameters for Dynamical System

In [3]:
```
import numpy as np

# Number of cores to be used for parallel computing
Ncores = 18

# Incompressible/Compressible flow. {True, False}
Incompressible = True

# Periodic boundary conditions
periodic_x = False
periodic_y = False
Periodic = [periodic_x, periodic_y]

## Compute Meshgrid
X, Y = np.meshgrid(x, y)

# List of parameters of the flow.
params_data = {"X": X, "Y": Y, "Time": time, "U": U, "V": V, "Ncores": Ncores,
               "Incompressible": Incompressible, "Periodic": Periodic}
```

Spatio-temporal domain of Dynamical System

In [4]:
```
%%time
# Initial time (in days)
t0 = 25

# Final time (in days)
tN = 45

# time step-size
dt = .2

time = np.arange(t0, tN+dt, dt)

# longitudinal and latitudinal boundaries (in degrees)
xmin = 0
xmax = 5
ymin = -35
ymax = -30

# spacing of meshgrid (in degrees)
dx = 0.025
dy = 0.025

x_domain = np.arange(xmin, xmax + dx, dx)
y_domain = np.arange(ymin, ymax + dy, dy)

X_domain, Y_domain = np.meshgrid(x_domain, y_domain)

params_DS = {"time": time, "X_domain": X_domain, "Y_domain": Y_domain}
```

```
CPU times: user 483 µs, sys: 397 µs, total: 880 µs
Wall time: 422 µs
```

Initialize Dynamical System

In [5]:
```
%%time
from ipynb.fs.defs.DynamicalSystem import *

DS = Dynamical_System(params_data, params_DS)
```

Velocity interpolation

In [6]:
```
%%time
# Interpolate velocity data using cubic spatial interpolation
DS._Interpolation_velocity("cubic")
```

```
CPU times: user 180 ms, sys: 18.3 ms, total: 198 ms
Wall time: 197 ms
```

Gradient of flow map over meshgrid of initial conditions

In [7]:
```
%%time
# aux_grid = True --> Use auxiliary grid for numerical computation of gradient.
# Otherwise aux_grid = False.
aux_grid = True

grad_Fmap_grid = DS._grad_Fmap_grid(aux_grid)
```

```
CPU times: user 13.1 s, sys: 5.53 s, total: 18.6 s
Wall time: 11min 16s
```

Polar Rotation Angle (PRA)

In [8]:
```
from ipynb.fs.defs.PRA import _PRA
PRA = _PRA(grad_Fmap_grid)
```

In [12]:
```
######################## PLOT RESULTS ########################
import matplotlib.pyplot as plt

# Figure/Axes
fig = plt.figure(figsize=(16, 10))
ax = plt.axes()

# Contourplot of TSE over meshgrid of initial conditions
cax = ax.contourf(X_domain, Y_domain, PRA, cmap = "gist_rainbow_r", levels = 600)

# Axis Labels
ax.set_xlabel("long (°)", fontsize = 16)
ax.set_ylabel("lat (°)", fontsize = 16)

# Ticks
ax.set_xticks(np.arange(np.min(X_domain), np.max(X_domain), 1))
ax.set_yticks(np.arange(np.min(Y_domain), np.max(Y_domain), 1))

# Colorbar
cbar = fig.colorbar(cax, ticks = np.linspace(0, 3, 7))
cbar.ax.set_ylabel(r'$(\dfrac{1}{d})$', rotation = 0, labelpad = 10, fontsize = 16)

# Title
ax.set_title(r'$ \mathrm{\overline{PRA}}$'+f'$_{{{int(time[0])}}}^{{{int(time[-1])}}}$', fontsize = 20)

plt.show()
```