Student_4_CMA_ES

April 14, 2021

1 Covariance Matrix Adaptive ES

Note this is a simplified version of the original algorithm. Make sure to check the original paper before applying this optimizer to a real-world problem!

[ ]:
```
%matplotlib notebook
from math import sin, cos, sqrt, pi
from matplotlib import cm
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
import matplotlib.transforms as transforms
```

[ ]:
```
def loss_function(x, a=10):
    dummy = a * len(x)
    for ii in range(len(x)):
        dummy += x[ii] ** 2 - a * cos(2 * pi * x[ii])
    return dummy
```

[ ]:
```
def plot_loss(ax):

    x = np.linspace(-5, 5, 200)
    y = np.linspace(-5, 5, 200)
    X, Y = np.meshgrid(x, y)

    Z = np.zeros_like(X)
    for ii in range(X.shape[0]):
        for jj in range(X.shape[1]):
            Z[ii][jj] = loss_function([X[ii][jj], Y[ii][jj]])
    img = ax.contour(X, Y, Z, levels=30, cmap=cm.coolwarm)
    plt.colorbar(img, ax=ax)

    return ax
```

[ ]:
```
def error_ellipse(ax, xc, yc, cov, sigma=3, **kwargs):
    """
    https://github.com/megbedell/plot_tools/blob/master/error_ellipse.py
    Plot an error ellipse contour over your data.
```

```
    Inputs:
    ax : matplotlib Axes() object
    xc : x-coordinate of ellipse center
    yc : x-coordinate of ellipse center
    cov : covariance matrix
    sigma : # sigma to plot (default 1)
    additional kwargs passed to matplotlib.patches.Ellipse()
    """
    w, v = np.linalg.eigh(cov)  # assumes symmetric matrix
    order = w.argsort()[::-1]
    w, v = w[order], v[:, order]
    theta = np.degrees(np.arctan2(*v[:, 0][::-1]))  # * unpacks argument␣
↪instead of [0]
    ellipse = Ellipse(
        xy=(xc, yc),
        width=2.0 * sigma * np.sqrt(w[0]),
        height=2.0 * sigma * np.sqrt(w[1]),
        angle=theta,
        **kwargs
    )
    ellipse.set_facecolor("none")
    ax.add_artist(ellipse)

    return ax
```

[ ]:
```
def covar_helper(x1, mu_x1, x2, mu_x2):
    if len(x1) != len(x2):
        raise Exception

    tmp = 0
    for ii in range(len(x1)):
        tmp += (x1[ii] - mu_x1) * (x2[ii] - mu_x2)

    return tmp / len(x1)
```

[ ]:
```
 # Simplified CMA ES
n_evolutions = 20
n_population = 100
best_perc = 0.25
start = [3.5, 3.5]
init_population = [start] * n_population
sig_0 = 0.65
population = np.random.normal(init_population, sig_0)

centers = [start]

# Inits for plot
```

```
fig = plt.figure()
ax = fig.gca()

axes = plt.gca()
axes.set_xlim([-5, 5])
axes.set_ylim([-5, 5])

plt.ion()
fig.show()
fig.canvas.draw()

plot_loss(ax)

for episode in range(n_evolutions):

    if episode > 0:
        # confidence_ellipse(cov, mean_old, ax)
        ax = error_ellipse(
            ax, mean_old[0], mean_old[1], cov, ec="green", zorder=9999
        )

    # Population Update Strategy

    # Todo: store the loss of each particle in a list named scores


    # Todo: Find the best best_perc percent particles with respect to their␣
↪loss.
    #       Store these particles in a numpy array named best_population


    # Todo: Find the mean of the best population and save it as mean


    centers.append(list(mean))

    if episode == 0:
        mean_old = mean

    sig_xx = covar_helper(
        best_population[:, 0], mean_old[0], best_population[:, 0], mean_old[0]
    )
    sig_xy = covar_helper(
        best_population[:, 0], mean_old[0], best_population[:, 1], mean_old[1]
    )
    sig_yy = covar_helper(
        best_population[:, 1], mean_old[1], best_population[:, 1], mean_old[1]
```

```
    )

    # Todo: build a covariance matrix with the entries above and draw a new␣
↪population around the mean calculated above.


    mean_old = mean


    # Plot population
    fig.canvas.draw()

centers = np.array(centers)

plt.plot(centers[:,0], centers[:,1], ".-g", label="Centers")
plt.plot(centers[0,0], centers[0,1], "or", label="Start")
plt.plot(centers[-1,0], centers[-1,1], "xr", label="End")
plt.legend()
```

[ ]:

```

```