

```

In [1]: 1 import numpy as np
        2 import pandas as pd
        3 import matplotlib.pyplot as plt
        4 %matplotlib inline

In [2]: 1 def f(x):
        2     return(100 * (x[1][0] - x[0][0] ** 2) ** 2 + (1 - x[0][0]) ** 2)
        3
        4 def grad_f(x):
        5     return(np.array([[400 * x[0][0] ** 3 - 400 * x[0][0] * x[1][0] + 2 * x[0][0] - 2], [200 * (x[1][0] - x[0][0] ** 2)]]))
        6
        7 def hessian_f(x):
        8     return(np.array([[1200 * x[0][0] ** 2 - 400 * x[1][0] + 2, -400 * x[0][0]], [-400 * x[0][0], 200]]))

In [3]: 1 def back_track(x, alpha, c, d, rho):
        2     while True:
        3         if f(x + alpha * d) <= f(x) + c * alpha * grad_f(x).T @ d:
        4             return alpha
        5         alpha *= rho

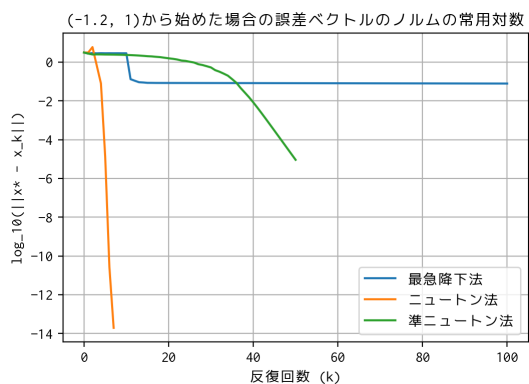
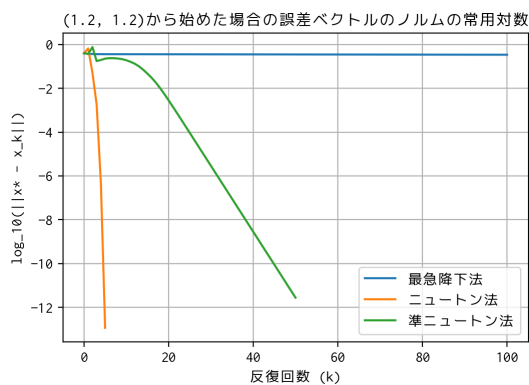
In [4]: 1 def gradient_descent(x_0, alpha = 0.5, c = 0.1, rho = 0.8, n_itr = 100):
        2     xs = np.array([x_0])
        3     for k in range(n_itr):
        4         d = -1 * grad_f(xs[k])
        5         xs = np.append(xs, [xs[k] + back_track(xs[k], alpha, c, d, rho) * d], axis = 0)
        6     return xs
        7
        8 def newton_method(x_0, n_itr = 50):
        9     xs = np.array([x_0])
        10    for k in range(n_itr):
        11        d = -1 * np.linalg.inv(hessian_f(xs[k])) @ grad_f(xs[k])
        12        xs = np.append(xs, [xs[k] + d], axis = 0)
        13    return xs
        14
        15 def quasi_newton_method(x_0, H = np.array([[1, 0], [0, 1]]), alpha = 0.5, c = 0.1, rho = 0.8, n_itr = 50):
        16    xs = np.array([x_0])
        17    for k in range(n_itr):
        18        d = -1 * H @ grad_f(xs[k])
        19        xs = np.append(xs, [xs[k] + back_track(xs[k], alpha, c, d, rho) * d], axis = 0)
        20        s = xs[k + 1] - xs[k]
        21        y = grad_f(xs[k + 1]) - grad_f(xs[k])
        22        H = (np.identity(2) - (s @ y.T) / (y.T @ s)) @ H @ (np.identity(2) - (y @ s.T) / (y.T @ s)) + (s @ s.T) / (s.T @ y)
        23    return xs

In [5]: 1 def error(v, optimum_point = np.array([[1], [1]])):
        2     return np.log10(np.linalg.norm(v - optimum_point))
        3
        4 def plot_errors(start, start_str):
        5     gradient_transition = gradient_descent(start)
        6     newton_transition = newton_method(start)
        7     quasi_newton_transition = quasi_newton_method(start)
        8
        9     plt.plot(np.arange(len(gradient_transition)), np.apply_along_axis(error, 1, gradient_transition).flatten(), label='最急降下法')
        10    plt.plot(np.arange(len(newton_transition)), np.apply_along_axis(error, 1, newton_transition).flatten(), label='ニュートン法')
        11    plt.plot(np.arange(len(quasi_newton_transition)), np.apply_along_axis(error, 1, quasi_newton_transition).flatten(), label='準ニュートン法')
        12    plt.legend()
        13    plt.title(start_str + "から始めた場合の誤差ベクトルのノルムの常用対数")
        14    plt.xlabel('反復回数 (k)')
        15    plt.ylabel('log_10(||x* - x_k||)')
        16    plt.grid()
        17    plt.show()

In [6]: 1 plot_errors(np.array([[1.2], [1.2]]), '(1.2, 1.2)')
        2 plot_errors(np.array([[1.2], [1.2]]), '(-1.2, 1)')

```

/Users/uedatomohiro/.pyenv/versions/anaconda3-5.1.0/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning: divide by zero encountered in log10



```

In [7]: 1 def plot_transitions(start, start_str):
2         plt.scatter([1], [1], label='最適点', color='red')
3
4         gradient_transition = gradient_descent(start)
5         newton_transition = newton_method(start)
6         quasi_newton_transition = quasi_newton_method(start)
7
8         plt.plot(gradient_transition[:, 0], gradient_transition[:, 1], label='最急降下法')
9         plt.plot(newton_transition[:, 0], newton_transition[:, 1], label='ニュートン法')
10        plt.plot(quasi_newton_transition[:, 0], quasi_newton_transition[:, 1], label='準ニュートン法')
11        plt.legend()
12        plt.title(start_str + "から始めた場合の誤差ベクトルのノルムの常用対数")
13        plt.xlabel('x_1')
14        plt.ylabel('x_2')
15        plt.grid()
16        plt.show()

```

```

In [8]: 1 plot_transitions(np.array([[1.2], [1.2]]), '(1.2, 1.2)')
2       plot_transitions(np.array([[-1.2], [1.]]), '(-1.2, 1)')

```

