// Implementation in Octave

```
gradApprox = (J(theta + EPSILON) - ...
             J(theta - EPSILON)) / (2 * EPSILON);
```

Agora, se $\Theta \in \mathbb{R}^n$

thetaVec = [Theta1(:); Theta2(:); ...]

dá o reshape into loko e:

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + \varepsilon, \theta_2, \ldots, \theta_n) - J(\theta_1 - \varepsilon, \theta_2, \ldots, \theta_n)}{2\varepsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2 + \varepsilon, \ldots, \theta_n) - J(\theta_1, \theta_2 - \varepsilon, \ldots, \theta_n)}{2\varepsilon}$$

$$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \ldots, \theta_n + \varepsilon) - J(\theta_1, \theta_2, \ldots, \theta_n - \varepsilon)}{2\varepsilon}$$

→

// in Octave:

```
for i = 1:n,
    thetaPlus = thetaVec;
    thetaPlus(i) = thetaPlus(i) + EPSILON;
    thetaMinus = thetaVec;
    thetaMinus(i) = thetaMinus(i) - EPSILON;
    gradApprox(i) = (J(thetaPlus) - ...
                     J(thetaMinus)) / (2 * EPSILON);
end for;
```

Check that gradApprox ≈ DVec.

// implementation note

- Implement BP to compute DVec
- Implement numerical gradient checking
- Make sure gradApprox ≈ DVec
- TURN OFF gradient checking, because it's very computationally costly.