**Recomendación por orden**

**1º Unroll – desenrollar – matriz a vector**

```
def unroll(x,theta):
        #Desenrollo x
        dx = np.array(x).ravel(order='F')
        #Desenrollo theta
        dt = np.array(t).ravel(order='F')
        #Uno los dos vectores
        return np.stack(dx,dt)
```

**2º Roll – enrollar – vector a matriz**

```
def roll(vector, y, ncaracteristicas):
        #Numero productos
        Nproductos = y.shape[0]
        #Numero usuarios
        Nusuarios = y.shape[1]
        #Cojo valores del vector desde el principio hasta el tamaño de x
        X =
np.reshape(a=vector[:nproductos*ncaracteristicas],newshape=[numproductos,numcaracteristi
cas], order="F")
        #Cojo valores del vector desde el anterior hasta el final
        thetha =
np.reshape(a=vector[nproductos*ncaracteristicas:],newshape=[numcaracteristicas,numusuari
os], order="F")
        return x,theta
```

**3º Coste**

```
def coste(params,y,r,nc):
        x, theta = roll(params, y, r, nc, lambda_param)
        resultado = x @ theta - y
        resultado = resultado * r
        resultado = np.power(resultado, 2)
        resultado = np.sum(np.sum(resultado / 2))
        resultado = resultado + ((lambda_param / * np.sum(np.power(theta, 2))) + (
        (lambda_param / 2) * np.sum(np.power(x, 2)))
```

**4º Gradiente**

```
def gradiente(params, y, r, num_caracteristicas, lambda_param)
        x, theta = roll(params, y, r, num_caracteristicas)
        resultado = x @ theta - y
        # Filtrado resultado = resultado * r
        x_grad = resultado @ theta.T
        x_grad_regu = (x_grad + (lambda_param * x))
        theta_grad = x.T @ resultado
```

```
    theta_grad_regu = (theta_grad + (lambda_param * theta))
    return unroll(x_grad_regu, theta_grad_regu)
```

**5º Normalización**

```
def normalizacion(num_productos, num_usuarios, r, y):
  # Inicialización con ceros de Ymean y Ynorm con dimensiones adecuadas
  ymean = np.zeros(([num_productos, 1]))
  ynorm = np.zeros(([num_productos, num_usuarios]))
  # Para cada ítem
  for i in range(num_productos):
    # (Buena idea Vectorizar el for)
    idx = np.where(r[i, :] == 1)[0]
    ymean[i] = y[i, idx].mean()
    ynorm[i, idx] = y[i, idx] - ymean[i]
  print("Matrix de los valores medios de Y normalizados: ", ynorm.mean())
  return ymean, ynorm
```

**6º MAIN**

**1º Nuevo usuario**

```
null_array = np.empty([num_productos, 1])
y = np.append(y, null_array, axis=1)
# Append siempre añade al final del array. Axis=1 para añadir los
valores como columna
ceros_valoraciones = np.zeros((num_productos, 1))
r = np.append(r, ceros_valoraciones, axis=1)
# Append siempre añade al final del array. Axis=1 para añadir los
valores como columna

num_usuarios = num_usuarios + 1 # IMPORTANTE!
```

**2º Randomizar x y theta**

```
x = np.random.rand(num_productos,num_caracteristicas)*(2*0.12)
thetha = np.random.rand(num_caracterisitcas,num_usuarios) * (2 * 0.12)
```

**3º Normalizar**

```
ymean, ynorm = normalizacion(num_productos, num_usuarios, r, y)
```

**4º Inicializar lambda**

```
lambda_param = 0.1
```

**5.- Desenrollamos x y theta en un vector**

```
params = unroll(x,theta)
```

**6.- Aplicamos el algoritmo de optimización**

```
fmin = opt.fmin_cg(maxiter=200, f=rcoste, x0=params, fprime=rgrad,
        args=(ynorm, r, num_caracteristicas, lambda_param))
```

**7.- Hacemos la predicción con normalización**

```
predictions = x @ theta + ymean
```

**8.- Cogemos las predicciones del último usuario, es decir, del nuevo usuario**

```
my_preds = predictions[:, -1]
```

**9.- Los mejores**

```
idx = np.argsort(my_preds, axis=0)[::-1]
```

**10.- Imprimir por pantalla**

```
print("Predicción de los 10 mejores productos para el usuario nuevo:")
for i in range(10):
    j = int(idx[i])
        print('Predicción de las valoración de {0} para el producto
        {1}.'.format(str(float(my_preds[j])), nombre_productos[j]))
```