ODE example

February 27, 2020

[2]:
```
import autograd.numpy as np
from autograd import grad
import autograd.numpy.random as npr

from autograd.core import primitive

from matplotlib import pyplot as plt
%matplotlib inline
```

[3]:
```
nx = 100
dx = 1. / nx
```

0.0.1 La ecuacion que vamos a resolver es la siguiente:

$$\frac{d\psi}{dx} + \left(x + \frac{1+3x^2}{1+x+x^3}\right) = x^3 + 2x + x^2\left(\frac{1+3x^2}{1+x+x^3}\right)$$

Con condiciones iniciales $\psi(0) = 1$

[4]:
```
def A(x):
    '''
        Left part of initial equation
    '''
    return x + (1. + 3.*x**2) / (1. + x + x**3)
```

$$\left(x + \frac{1+3x^2}{1+x+x^3}\right)$$

[4]:
```
def B(x):
    '''
        Right part of initial equation
    '''
    return x**3 + 2.*x + x**2 * ((1. + 3.*x**2) / (1. + x + x**3))
```

$$x^3 + 2x + x^2\left(\frac{1+3x^2}{1+x+x^3}\right)$$

[5]:
```python
def f(x, psy):
    '''
        d(psy)/dx = f(x, psy)
        This is f() function on the right
    '''
    return B(x) - psy * A(x)
```

$$\frac{d\psi}{dx} = f(x, \psi)$$

[6]:
```python
def psy_analytic(x):
    '''
        Analytical solution of current problem
    '''
    return (np.exp((-x**2)/2.)) / (1. + x + x**3) + x**2
```

$$\frac{e^{\frac{-x^2}{2}}}{1 + x + x^3} + x^2$$

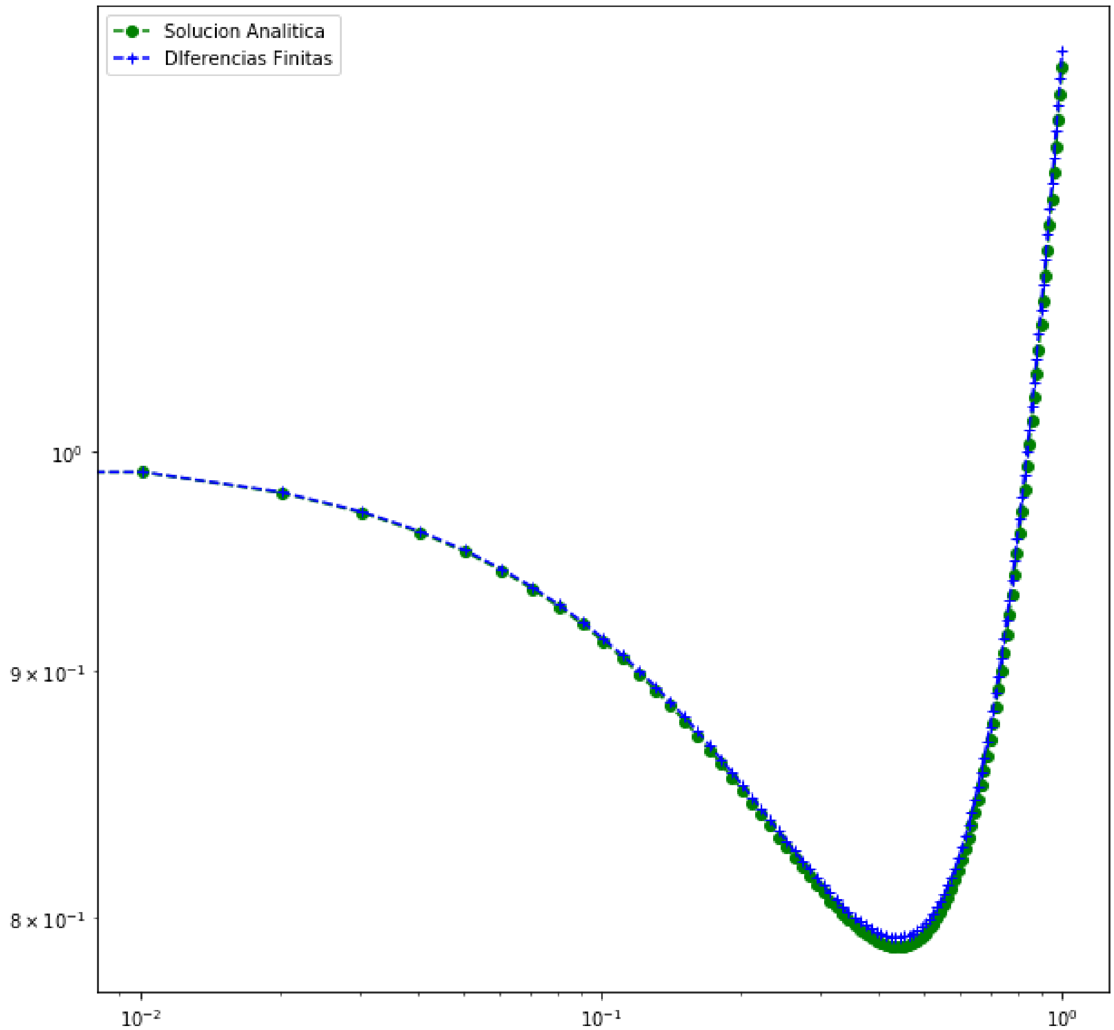
0.0.2 Primero lo resolvemos con el metodo de diferencias finitas

[7]:
```python
x_space = np.linspace(0, 1, nx)
y_space = psy_analytic(x_space)
psy_fd = np.zeros_like(y_space)
psy_fd[0] = 1. # IC

for i in range(1, len(x_space)):
    psy_fd[i] = psy_fd[i-1] + B(x_space[i]) * dx - psy_fd[i-1] * A(x_space[i])␣
 ↪* dx

fig, ax = plt.subplots(figsize=(10, 10))
ax.plot(x_space, y_space, color='green', marker='o', linestyle='dashed',␣
 ↪label='Solucion Analitica')
ax.plot(x_space, psy_fd, color='blue', marker='+', linestyle='dashed',␣
 ↪label='DIferencias Finitas')
ax.set_xscale('log')
ax.set_yscale('log')
ax.legend()
```

[7]:
```
<matplotlib.legend.Legend at 0x7f7bd0048bd0>
```



0.0.3 Ahora creamos las funciones de la red neuronal

[8]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

[9]:
```
def sigmoid_grad(x):
    return sigmoid(x) * (1 - sigmoid(x))
```

$$\frac{d\sigma}{dx} = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x) * (1 - \sigma(x))$$

[10]:
```python
def neural_network(W, x):
    a1 = sigmoid(np.dot(x, W[0]))
    return np.dot(a1, W[1])
```

$$a1 = \sigma(z_i) = \sigma(\sum_{j=1}^{n} w_{ji}x_j)$$

$$N(x, p) = \sum_{j=1}^{n} v_j \sigma(z_i)$$

[11]:
```python
def d_neural_network_dx(W, x, k=1):
    return np.dot(np.dot(W[1].T, W[0].T**k), sigmoid_grad(x))
```

$$\frac{d^k N}{dx^k} = \sum_{j=1}^{m} v_j w_j^k \sigma_j^{(k)}$$

[15]:
```python
def loss_function(W, x):
    loss_sum = 0.
    for xi in x:
        net_out = neural_network(W, xi)[0][0]
        psy_t = 1. + xi * net_out
        d_net_out = d_neural_network_dx(W, xi)[0][0]
        d_psy_t = net_out + xi * d_net_out
        func = f(xi, psy_t)
        err_sqr = (d_psy_t - func)**2

        loss_sum += err_sqr
    return loss_sum
```

$$\psi_t(x) = A + xN(x, p)$$

$$\frac{d\psi_t(x_i)}{dx} = N(x, p) + \left(x_i \frac{dN(x, p)}{dt} \right)$$

$$\mathrm{E}[\overrightarrow{p}] = \sum_i \left(\frac{d\psi_t(x_i)}{dx} - f(x_i, \psi_t(x_i)) \right)^2$$

$$$$

[21]:
```python
W = [npr.randn(1, 10), npr.randn(10, 1)]
lmb = 0.001

#x = np.array(1)
```

```
#print (neural_network(W, x))
#print (d_neural_network_dx(W, x))

for i in range(1000):
    loss_grad =  grad(loss_function)(W, x_space)

    #print (loss_grad[0].shape, W[0].shape)
    #print (loss_grad[1].shape, W[1].shape)
    #Descenso de gradiente
    W[0] = W[0] - lmb * loss_grad[0]
    W[1] = W[1] - lmb * loss_grad[1]

    #print (loss_function(W, x_space))
```

```
/home/david/anaconda3/lib/python3.7/site-
packages/autograd/numpy/numpy_vjps.py:53: RuntimeWarning: overflow encountered
in square
  lambda ans, x, y : unbroadcast_f(y, lambda g: - g * x / y**2))
/home/david/anaconda3/lib/python3.7/site-packages/autograd/tracer.py:48:
RuntimeWarning: overflow encountered in exp
  return f_raw(*args, **kwargs)
/home/david/anaconda3/lib/python3.7/site-
packages/autograd/numpy/numpy_vjps.py:75: RuntimeWarning: invalid value
encountered in multiply
  defvjp(anp.exp,    lambda ans, x : lambda g: ans * g)
```

[17]:
```
print(loss_function(W, x_space))
```

```
6.604535158614862
```

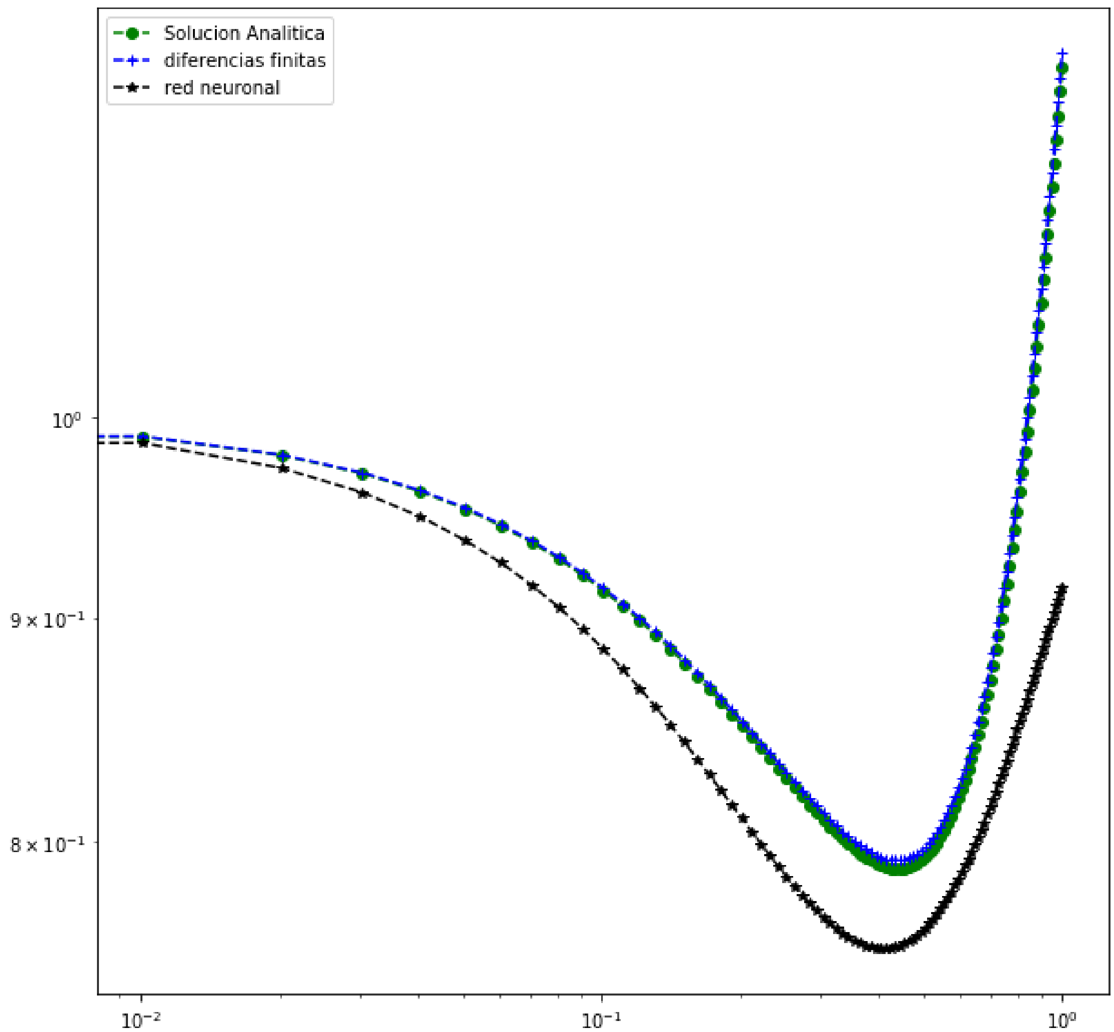
[18]:
```
print (W)
```

```
[array([[-2.1971797 , -2.58612512, -2.34397705, -2.4398557 , -2.86451077,
        -2.46400438, -2.73147661, -2.17382848, -2.76517548, -1.42970799]]),
array([[ 0.17102679],
       [-1.37328375],
       [-0.61107635],
       [-0.28843102],
       [-0.01831923],
       [ 0.63170298],
       [-0.90038194],
       [ 0.19777828],
       [-1.11511813],
       [ 0.63530697]])]
```

[19]:
```
res = [1 + xi * neural_network(W, xi)[0][0] for xi in x_space]
```

[20]:
```
fig, ax = plt.subplots(figsize=(10, 10))
ax.plot(x_space, y_space, color='green', marker='o', linestyle='dashed',␣
↪label='Solucion Analitica')
ax.plot(x_space, psy_fd, color='blue', marker='+', linestyle='dashed',␣
↪label='diferencias finitas')
ax.plot(x_space, res, color='black', marker='*', linestyle='dashed', label='red␣
↪neuronal')
ax.set_xscale('log')
ax.set_yscale('log')
ax.legend()
```

[20]: `<matplotlib.legend.Legend at 0x7f7ba8853890>`



[ ]:

```
[ ]:
```

```
[ ]:
```