In [0]:
```python
import numpy as np
import math
import random
```

In [2]:
```python
patterns = []
classes = []

filename = 'Iris_data.txt'
file = open(filename,'r')

for line in file.readlines():
    row = line.strip().split(',')
    patterns.append(row[0:4])
    classes.append(row[4])
print("Iris Data Loaded")
file.close
```

```
Iris Data Loaded
```

Out[2]:
```
<function TextIOWrapper.close>
```

In [3]:
```python
patterns = np.asarray(patterns,dtype=np.float32)
sample_no = np.random.randint(0,len(patterns))

print("Sample pattern: " + str(patterns[int(sample_no)]))
print("Class of the above pattern: " + str(classes[int(sample_no)]))
```

```
Sample pattern: [5.4 3.4 1.5 0.4]
Class of the above pattern: Iris-setosa
```

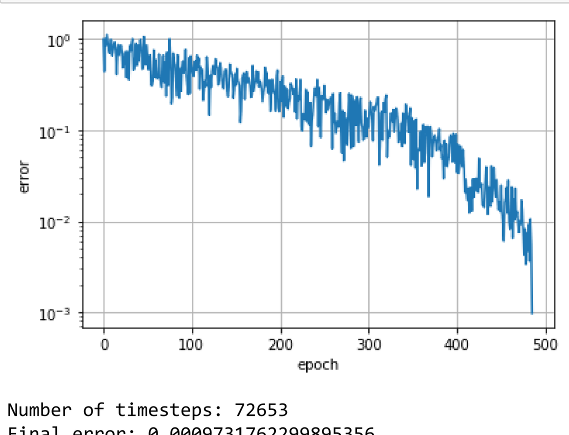
In [4]:
```python
def mapunits(input_len,size='small'):
    heuristic_map_units = 5*input_len**0.54321
    if size == 'big':
        heuristic_map_units = 4*(heuristic_map_units)
    else:
        heuristic_map_units = 0.25*(heuristic_map_units)
    return heuristic_map_units

map_units = mapunits(len(patterns),size='big')
print("Calcular el numero de mapunits de forma heuristica: " + str(int(map_units)))
```

```
Calcular el numero de mapunits de forma heuristica: 304
```

In [5]:
```python
import matplotlib.pyplot as plt
%matplotlib inline

def Eucli_dists(MAP,x):
    x = x.reshape((1,1,-1))
    #print(x)
    Eucli_MAP = MAP - x
    Eucli_MAP = Eucli_MAP**2
    Eucli_MAP = np.sqrt(np.sum(Eucli_MAP,2))
    return Eucli_MAP

input_dimensions = 4

map_width = 9
map_height = 5
MAP = np.random.uniform(size=(map_height,map_width,input_dimensions))
prev_MAP = np.zeros((map_height,map_width,input_dimensions))

radius0 = max(map_width,map_height)/2
learning_rate0 = 0.1

coordinate_map = np.zeros([map_height,map_width,2],dtype=np.int32)

for i in range(0,map_height):
    for j in range(0,map_width):
        coordinate_map[i][j] = [i,j]

epochs = 500
radius=radius0
learning_rate = learning_rate0
max_iterations = len(patterns)+1
too_many_iterations = 10*max_iterations

convergence = [1]

timestep=1
e=0.001
flag=0

epoch=0
while epoch<epochs:

    shuffle = np.random.randint(len(patterns), size=len(patterns))
    for i in range(len(patterns)):

        # difference between prev_MAP and MAP
        J = np.linalg.norm(MAP - prev_MAP)
        #print(J)
        # J = || euclidean distance between previous MAP and current MAP  ||

        if  J <= e: #if converged (convergence criteria)
            flag=1
            break

        else:

            #if timestep == max_iterations and timestep != too_many_iterations:
            #    epochs += 1
            #    max_iterations = epochs*len(patterns)

            pattern = patterns[shuffle[i]]
            pattern_ary = np.tile(pattern, (map_height, map_width, 1))
            Eucli_MAP = np.linalg.norm(pattern_ary - MAP, axis=2)

            # Get the best matching unit(BMU) which is the one with the smallest Euclidean distance
            BMU = np.unravel_index(np.argmin(Eucli_MAP, axis=None), Eucli_MAP.shape)
            #BMU[1] = np.argmin(Eucli_MAP, 1)[int(BMU[0])]

            #Eucli_from_BMU = Eucli_dists(coordinate_map,BMU)

            prev_MAP = np.copy(MAP)

            for i in range(map_height):
                for j in range(map_width):
                    distance = np.linalg.norm([i - BMU[0], j - BMU[1]])
                    if distance <= radius:
                        #theta = math.exp(-(distance**2)/(2*(radius**2)))
                        MAP[i][j] = MAP[i][j] + learning_rate*(pattern-MAP[i][j])

            learning_rate = learning_rate0*(1-(epoch/epochs))
            #time_constant = max_iterations/math.log(radius)
            radius = radius0*math.exp(-epoch/epochs)
            #print([learning_rate, radius])

            timestep+=1

    if J < min(convergence):
        print('Lower error found: %s' %str(J) + ' at epoch: %s' % str(epoch))
        print('\tLearning rate: ' + str(learning_rate))
        print('\tNeighbourhood radius: ' + str(radius))
        MAP_final = MAP
    convergence.append(J)

    if flag==1:
        break
    epoch+=1
```

```
Lower error found: 0.4405964433891063 at epoch: 0
	Learning rate: 0.1
	Neighbourhood radius: 4.5
Lower error found: 0.409563001432001 at epoch: 14
	Learning rate: 0.09720000000000001
	Neighbourhood radius: 4.375747650605611
Lower error found: 0.3798619591252999 at epoch: 25
	Learning rate: 0.095
	Neighbourhood radius: 4.280532410253213
Lower error found: 0.35941816426896067 at epoch: 28
	Learning rate: 0.0944
	Neighbourhood radius: 4.254926111506784
Lower error found: 0.31497606936753625 at epoch: 53
	Learning rate: 0.08940000000000001
	Neighbourhood radius: 4.047410916341658
Lower error found: 0.3088356054016895 at epoch: 54
	Learning rate: 0.0892
	Neighbourhood radius: 4.039324183936957
Lower error found: 0.2948401899968131 at epoch: 65
	Learning rate: 0.08700000000000001
	Neighbourhood radius: 3.951429439142526
Lower error found: 0.23667878192530656 at epoch: 70
	Learning rate: 0.08600000000000001
	Neighbourhood radius: 3.9121120592946266
Lower error found: 0.19422292493954424 at epoch: 76
	Learning rate: 0.0848
	Neighbourhood radius: 3.8654472633350556
Lower error found: 0.1460110611674211 at epoch: 119
	Learning rate: 0.0762
	Neighbourhood radius: 3.546912109921967
Lower error found: 0.1216733555222412 at epoch: 154
	Learning rate: 0.0692
	Neighbourhood radius: 3.3071189310408093
Lower error found: 0.10948147280587228 at epoch: 223
	Learning rate: 0.055400000000000005
	Neighbourhood radius: 2.880826974277412
Lower error found: 0.09877907325888859 at epoch: 224
	Learning rate: 0.055200000000000006
	Neighbourhood radius: 2.8750710781436233
Lower error found: 0.06658740026863745 at epoch: 234
	Learning rate: 0.053200000000000004
	Neighbourhood radius: 2.8181408564590016
Lower error found: 0.06252039400446609 at epoch: 258
	Learning rate: 0.0484
	Neighbourhood radius: 2.686065267034611
Lower error found: 0.05623650254531298 at epoch: 269
	Learning rate: 0.0462
	Neighbourhood radius: 2.627617118186829
Lower error found: 0.04614562785283791 at epoch: 271
	Learning rate: 0.0458
	Neighbourhood radius: 2.617127642651117
Lower error found: 0.0415913095517137 at epoch: 311
	Learning rate: 0.0378
	Neighbourhood radius: 2.415913307511568
Lower error found: 0.04125885806603384 at epoch: 347
	Learning rate: 0.030600000000000006
	Neighbourhood radius: 2.2480819742409524
Lower error found: 0.03874098197968329 at epoch: 352
	Learning rate: 0.029600000000000005
	Neighbourhood radius: 2.2257131848517564
Lower error found: 0.022632350657893374 at epoch: 354
	Learning rate: 0.029200000000000004
	Neighbourhood radius: 2.2168281141006094
Lower error found: 0.018662435134190757 at epoch: 367
	Learning rate: 0.026600000000000002
	Neighbourhood radius: 2.159933419199412
Lower error found: 0.017028975277585297 at epoch: 411
	Learning rate: 0.017800000000000007
	Neighbourhood radius: 1.9779825216300644
Lower error found: 0.012422516169191246 at epoch: 413
	Learning rate: 0.017400000000000006
	Neighbourhood radius: 1.9700863943263185
Lower error found: 0.011930352087372066 at epoch: 434
	Learning rate: 0.013200000000000002
	Neighbourhood radius: 1.8890563086365142
Lower error found: 0.009331692850221784 at epoch: 451
	Learning rate: 0.009799999999999998
	Neighbourhood radius: 1.8259079985837694
Lower error found: 0.006044070534305374 at epoch: 452
	Learning rate: 0.009599999999999997
	Neighbourhood radius: 1.822259831969272
Lower error found: 0.004260140601381133 at epoch: 475
	Learning rate: 0.0050000000000000044
	Neighbourhood radius: 1.7403346055452555
Lower error found: 0.003353256139338615 at epoch: 477
	Learning rate: 0.004600000000000004
	Neighbourhood radius: 1.7333871712548985
Lower error found: 0.0009731762299895356 at epoch: 484
	Learning rate: 0.003200000000000003
	Neighbourhood radius: 1.7092888328312044
```

In [6]:
```python
plt.plot(convergence)
plt.ylabel('error')
plt.xlabel('epoch')
plt.grid(True)
plt.yscale('log')
plt.show()
print('Number of timesteps: ' + str(timestep))
print('Final error: ' + str(J))
```




```
Number of timesteps: 72653
Final error: 0.0009731762299895356
```

In [7]:
```python
BMU = np.zeros([2],dtype=np.int32)
result_map = np.zeros([map_height,map_width,3],dtype=np.float32)

i=0
for pattern in patterns:

    pattern_ary = np.tile(pattern, (map_height, map_width, 1))
    Eucli_MAP = np.linalg.norm(pattern_ary - MAP_final, axis=2)

    # Get the best matching unit(BMU) which is the one with the smallest Euclidean distance
    BMU = np.unravel_index(np.argmin(Eucli_MAP, axis=None), Eucli_MAP.shape)

    x = BMU[0]
    y = BMU[1]

    if classes[i] == 'Iris-setosa':
        if result_map[x][y][0] <= 0.5:
            result_map[x][y] += np.asarray([0.5,0,0])
    elif classes[i] == 'Iris-virginica':
        if result_map[x][y][1] <= 0.5:
            result_map[x][y] += np.asarray([0,0.5,0])
    elif classes[i] == 'Iris-versicolor':
        if result_map[x][y][2] <= 0.5:
            result_map[x][y] += np.asarray([0,0,0.5])
    i+=1
result_map = np.flip(result_map,0)

#print result_map

print("Red = Iris-Setosa")
print("Blue = Iris-Virginica")
print("Green = Iris-Versicolor")

plt.imshow(result_map, interpolation='nearest')
```

```
Red = Iris-Setosa
Blue = Iris-Virginica
Green = Iris-Versicolor
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x7fb333b63ac8>
```