Time Series Analysis and Recurrent Neural Network
Giacomo Barzon - 3626438
Exercise 6

December 4, 2019

1 Task 1 - Poisson latent variable models

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat                    #import mat data
import scipy as sp
```

In [2]:
```
# Load data
mat_file = loadmat('ex6file.mat')
#print(mat_file)

# u: external forces
u = mat_file['u']

# model matrices
A = mat_file['A']
B = mat_file['B']
Gamma = mat_file['Gamma']
Sigma = mat_file['Sigma']

# initial conditions
nu0 = mat_file['n0'].reshape(-1)
mu0 = mat_file['mu0'].reshape(-1)

# set random seed
np.random.seed(1)
```

1.0.1 1) Create time series of M × T (M = 2, T = 100) dimensional latent states Z = {zt} (named 'z'), and N × T dimensional observations C = {ct} (named 'c') from these variables and parameter settings

In [3]:
```
def latent_states(A, B, Sigma, u, mu0):
    n = u.shape[0]
    T = u.shape[1]
```

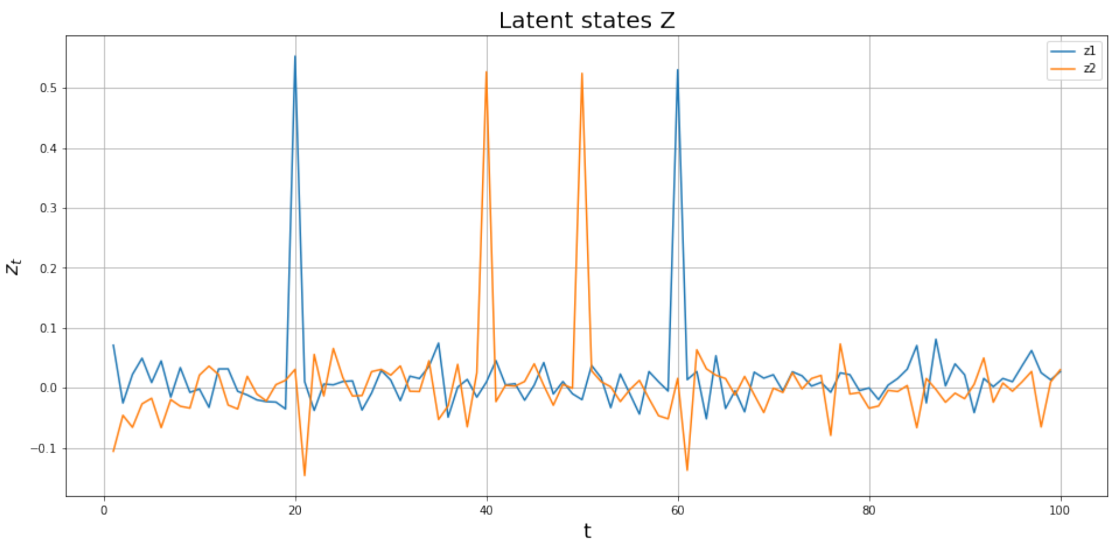
```
    z = np.zeros(( n, T ))

    # set first element
    z[:,0] = np.random.multivariate_normal(mu0, Sigma)

    # create vector of random generated values
    eps = np.random.multivariate_normal([0.,0.], Sigma, T-1).T

    for i in range(1,T):
        z[:,i] = A@z[:,i-1] + B@u[:,i] + eps[:,i-1]

    return z

def lmnd(Gamma, nu0, zi):
    return np.exp( np.log(nu0) + Gamma@zi )

def observations(Gamma, nu0, z):
    M = Gamma.shape[0]
    T = z.shape[1]

    c = np.zeros(( M,T ))

    for i in range(T):
        c[:,i] = np.random.poisson( lmnd(Gamma, nu0, z[:,i]) )

    return c
```

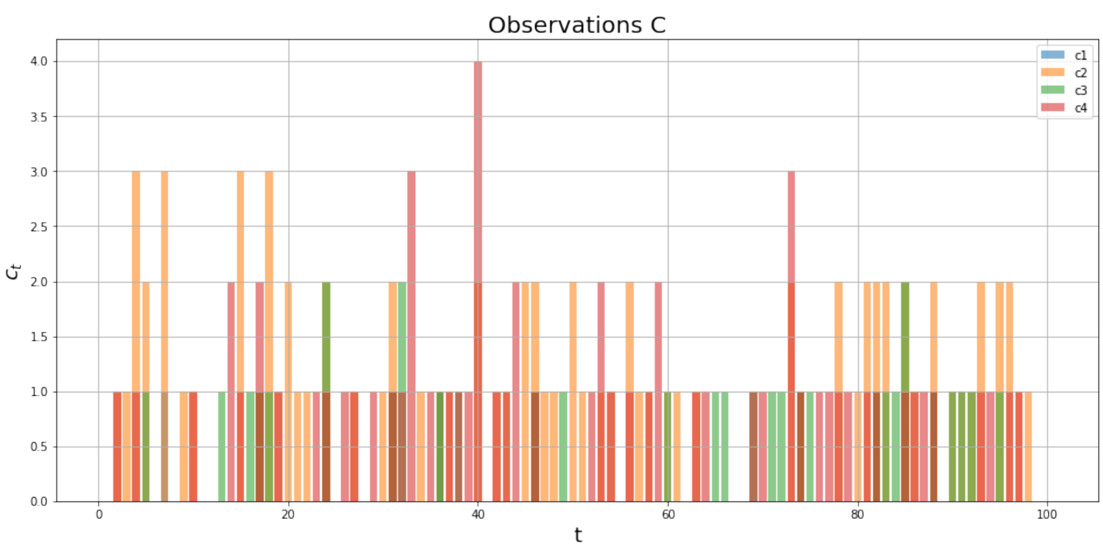
In [4]:
```
z = latent_states(A, B, Sigma, u, mu0)
c = observations(Gamma, nu0, z)
```

1.0.2 Plot them

In [5]:
```
# Plot the time series
fig1 = plt.subplots(figsize=[16,16])

plt.subplot(2,1,1)

for i in range(z.shape[0]):
    plt.plot(np.arange(z.shape[1])+1, z[i], label='z%i' %(i+1) )
plt.xlabel('t', fontsize = 18)
plt.ylabel(r'$z_t$', fontsize = 18)
plt.grid()
plt.legend()
plt.title('Latent states Z', fontsize = 20)

plt.subplot(2,1,2)
for i in range(c.shape[0]):
```

```
    plt.bar(np.arange(c.shape[1])+1, c[i], label='c%i' %(i+1) , alpha=0.55 )
plt.xlabel('t', fontsize = 18)
plt.ylabel(r'$c_t$', fontsize = 18)
plt.grid()
plt.legend()
plt.title('Observations C', fontsize = 20)

plt.show(fig1)
```





1.0.3 2) What is the joint data log-likelihood $log\ p(\{\vec{c}_t\vec{z}_t\}|\theta)$ of your generated time series?

$$log\ p(\{\vec{c}_t\vec{z}_t\}|\theta) = log\ p(\{\vec{c}_t\}|\{\vec{z}_t\},\theta) + log\ p(\{\vec{z}_t\}|\theta)$$

$$p(\{\vec{c}_t\}|\{\vec{z}_t\},\theta) \sim \ Poisson(\mu_t)$$
$$p(\{\vec{z}_t\}|\theta) \sim N(Az_{t-1} + Bu_t, \Sigma)$$
$$p(\{\vec{z}_1\}|\theta) \sim N(\mu_0, \Sigma)$$

```
In [6]: # compute log-likelihood
        def log_likelihood(A, B, Gamma, c, z, u, mu0, nu0):
            n = z.shape[0]
            T = z.shape[1]

            # log p(c|z)
            ll1 = 0.
            for i in range(T):
                lmbd = lmnd(Gamma, nu0, z[:,i])
                ll1 += c[:,i] * np.log(lmbd) - lmbd - sp.special.gammaln(c[:,i]+1)

            ll1 = sum(ll1)

            # log p(zt)
            ll2 = 0.
            for i in range(1,T):
                ll2 += (z[:,i] - A@z[:,i-1] -
                - B@u[:,i]).T @ np.linalg.inv(Sigma) @ (z[:,i] - A@z[:,i-1] - B@u[:,i])

            det = np.linalg.det(Sigma)
            ll2 += T * ( n*np.log(2.*np.pi) + np.log(np.abs(det)) )
            ll2 = -0.5 * ll2

            # log p(z1)
            ll3 = - 0.5 * (z[:,0] - mu0).T @  np.linalg.inv(Sigma) @ (z[:,0] - mu0)

            return ll1 + ll2 + ll3
```

```
In [7]: ll = log_likelihood(A, B, Gamma, c, z, u, mu0, nu0)
        print('Joint data log-likelihood:', ll)
```

```
Joint data log-likelihood: 119.15395813136317
```

2 Task 2 - Fixed points, stability, and bifurcations

Consider the univariate nonlinear map:

$$x_{t+1} = f(x_t, w, \theta) = w \cdot \sigma(x_t) + \theta \quad with\ \sigma(x) = \frac{1}{1 + exp(-x)}$$

2.0.1 1) For w = 8 and $\theta = -3.5$, find the fixed points of the system

Fixed point: $x^\star = f(x^\star) = w \cdot \frac{1}{1 + exp(-x^\star)} + \theta$

```
In [8]: # define parameters
        w = 8.
        theta = -3.5

        # define function f(x,w,theta)
        def map(x, w = w, theta = theta):
            return w / (1. + np.exp(-x)) + theta
```

```
In [9]: # find fixed points numerically
        def fixed_points(x, fx, tolerance=1e-2):
            x_star = np.abs(x - fx)

            # find local minima
            index1 = np.r_[True, x_star[1:] < x_star[:-1]] &
            np.r_[x_star[:-1] < x_star[1:], True]

            # find similar values (up to tolerance)
            index2 = x_star <= tolerance

            return x[index1 & index2]
```

```
In [10]: # define x array
         eps = 1e-5
         x = np.arange(-7.,7,eps)

         # compute f(x)
         fx = map(x)

         # find fixed points
         x_star = fixed_points(x, fx)
         print('Fixed points: ', x_star)
```

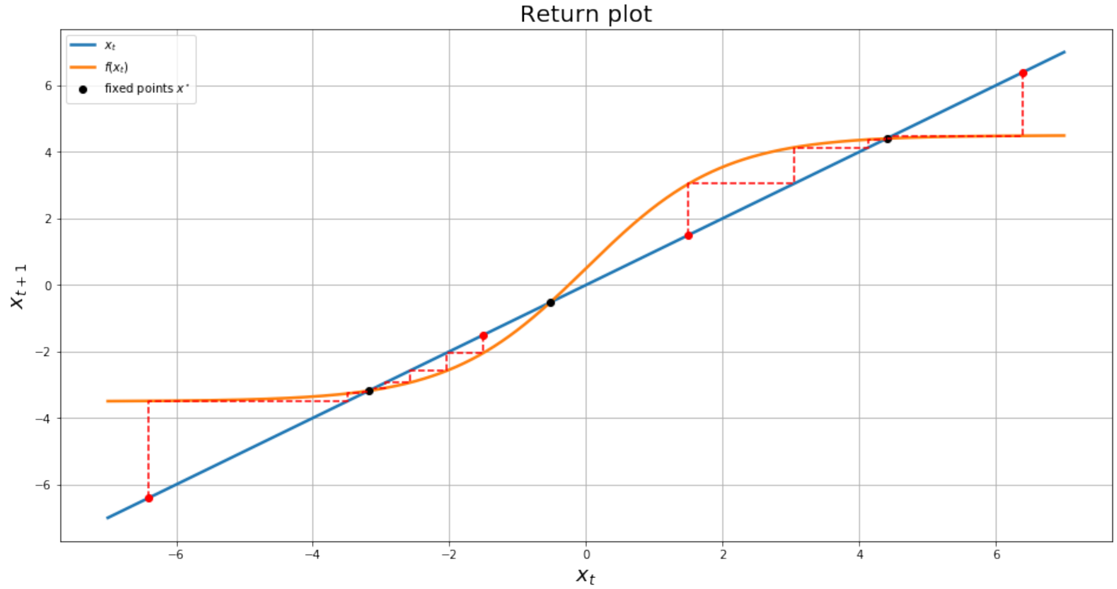
```
Fixed points:  [-3.18086 -0.52324  4.40329]
```

2.0.2 Visualize these in a graph

```
In [11]: # compute cobweb for fixed initial point
         def cobweb(x0, w = w, theta = theta, steps = 10):
             p = np.zeros(( 2, steps ))

             p[:,0] = [x0, x0]
             p[:,1] = [x0, map(x0, w = w, theta = theta) ]
```

```
    for i in range(2, steps, 2):
        p[:,i] = [ p[1,i-1], p[1,i-1] ]
        p[:,i+1] = [ p[0,i], map(p[0,i], w = w, theta = theta) ]

    return p
```

In [12]:
```
# plot return plot
fig1 = plt.subplots(figsize=[16,8])

#start_cob = [ -6, -1.5, 1.5, 6 ]
#cob = ...

line1 = cobweb(-6.4)
line2 = cobweb(-1.5)
line3 = cobweb(1.5)
line4 = cobweb(6.4)

plt.plot(x, x, label=r'$x_t$', linewidth=2.5 )
plt.plot(x, fx, label=r'$f(x_t)$', linewidth=2.5 )
plt.plot(x_star, x_star, 'ok' , label=r'fixed points $x^{\star}$', linewidth=3)

plt.plot(line1[0], line1[1], '--r')
plt.plot(line1[0,0], line1[1,0], 'or')
plt.plot(line2[0], line2[1], '--r')
plt.plot(line2[0,0], line2[1,0], 'or')
plt.plot(line3[0], line3[1], '--r')
plt.plot(line3[0,0], line3[1,0], 'or')
plt.plot(line4[0], line4[1], '--r')
plt.plot(line4[0,0], line4[1,0], 'or')

plt.xlabel(r'$x_{t}$', fontsize = 18)
plt.ylabel(r'$x_{t+1}$', fontsize = 18)
plt.grid()
plt.legend()
plt.title('Return plot', fontsize = 20)
plt.show()
```



2.0.3 Are they stable?

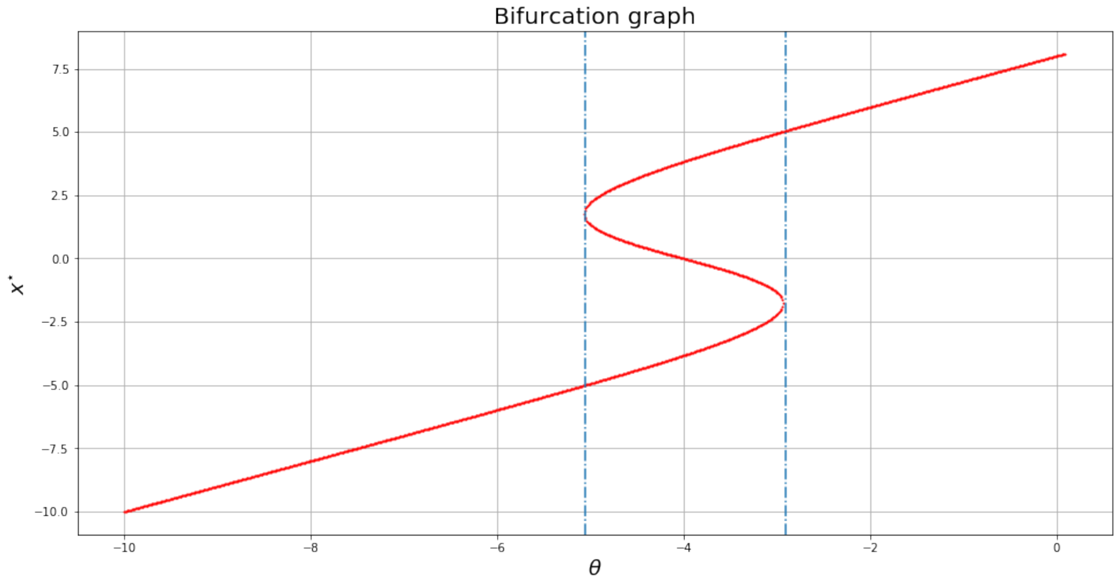
We can notice that fixing some starting points, the iterative map goes toward the two fixed points on the side, so we can deduce that they are stable. On the other hand none of them goes to the fixed point in the middle, so we can deduce that it's an unstable fixed point.

2.0.4 2) For w = 8, plot the bifurcation graph as a function of $\theta \in [-10\ 0]$. Include both stable and unstable objects

In [13]:
```
# range of thethas
thetas = np.arange(-10., 0.1, 0.01)

eps = 5e-5
x = np.arange(-10.,10, eps)

w = 8.

# compute fixed points for different thetas
bifurcation = []

for i in thetas:
    fx = map(x, w = w, theta = i)
    bifurcation.append( fixed_points(x,fx) )
```

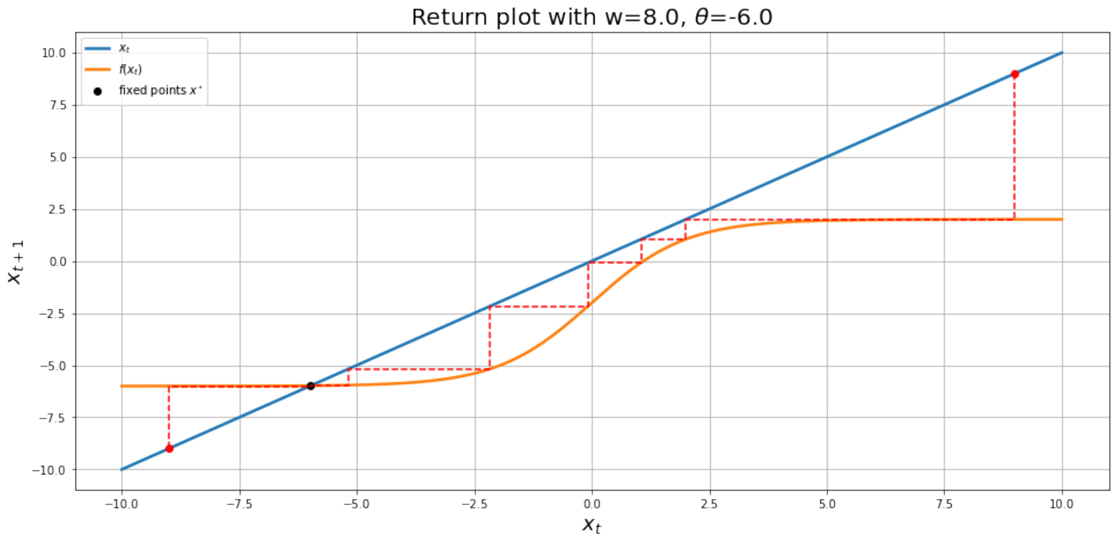
In [14]:
```
# plot bifurcation graph
fig1 = plt.subplots(figsize=[16,8])

for i in range(len(thetas)):
```

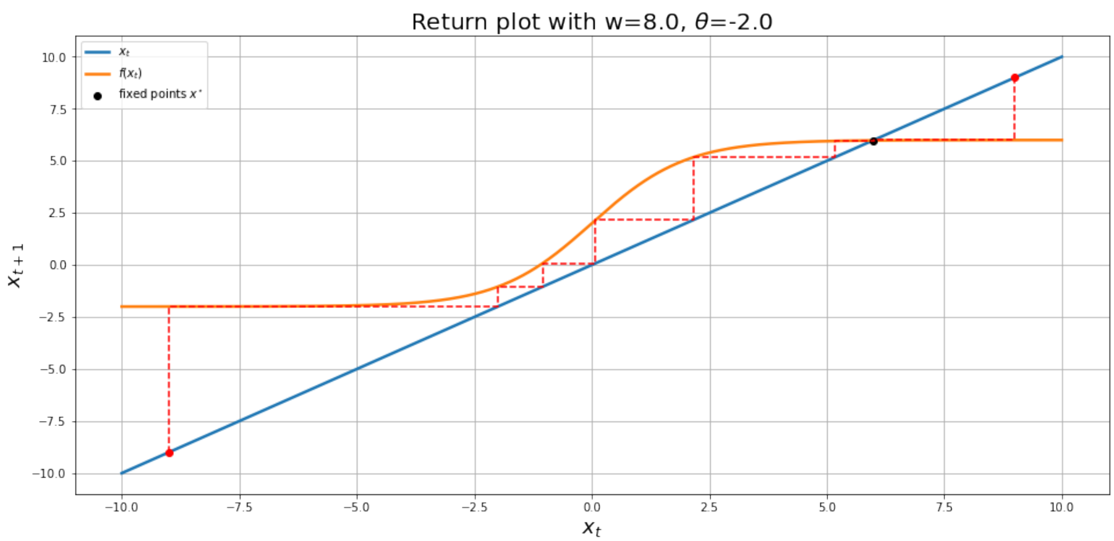
```
    plt.plot(np.zeros(bifurcation[i].shape)+thetas[i], bifurcation[i], '.r', markersize

plt.axvline(x = -5.06, ls='-.')
plt.axvline(x = -2.9, ls='-.')

plt.xlabel(r'$\theta$', fontsize = 18)
plt.ylabel(r'$x^{\star}$', fontsize = 18)
plt.grid()
plt.title('Bifurcation graph', fontsize = 20)
plt.show()
```




```
In [15]: # Compute fixed points for two values of theta as example
         theta1 = -6.
         theta2 = -2.

         x00 = -9
         x01 = 9

         fx1 = map(x, w = w, theta = theta1 )
         fp1 = fixed_points(x, fx1)
         line11 = cobweb(x00, w = w, theta = theta1 )
         line12 = cobweb(x01, w = w, theta = theta1, steps = 20 )

         fx2 = map(x, w = w, theta = theta2 )
         fp2 = fixed_points(x, fx2)
         line21 = cobweb(x00, w = w, theta = theta2, steps = 20 )
         line22 = cobweb(x01, w = w, theta = theta2 )
```

```python
# Plot the return plot
fig1 = plt.subplots(figsize=[16,16])

plt.subplot(2,1,1)

plt.plot(x, x, label=r'$x_t$', linewidth=2.5 )
plt.plot(x, fx1, label=r'$f(x_t)$', linewidth=2.5 )
plt.plot(fp1, fp1, 'ok', label=r'fixed points $x^{\star}$', linewidth=3)
plt.plot(line11[0], line11[1], '--r')
plt.plot(line11[0,0], line11[1,0], 'or')
plt.plot(line12[0], line12[1], '--r')
plt.plot(line12[0,0], line12[1,0], 'or')
plt.xlabel(r'$x_{t}$', fontsize = 18)
plt.ylabel(r'$x_{t+1}$', fontsize = 18)
plt.grid()
plt.legend()
plt.title(r'Return plot with w=%.1f, $\theta$=%.1f' %(w, theta1 ) , fontsize = 20)

plt.subplot(2,1,2)

plt.plot(x, x, label=r'$x_t$', linewidth=2.5 )
plt.plot(x, fx2, label=r'$f(x_t)$', linewidth=2.5 )
plt.plot(fp2, fp2, 'ok', label=r'fixed points $x^{\star}$', linewidth=3)
plt.plot(line21[0,0], line21[1,0], 'or')
plt.plot(line21[0], line21[1], '--r')
plt.plot(line22[0,0], line22[1,0], 'or')
plt.plot(line22[0], line22[1], '--r')
plt.xlabel(r'$x_{t}$', fontsize = 18)
plt.ylabel(r'$x_{t+1}$', fontsize = 18)
plt.grid()
plt.legend()
plt.title(r'Return plot with w=%.1f, $\theta$=%.1f' %(w, theta2 ), fontsize = 20)

plt.show(fig1)
```





2.0.5 How does the system change its dynamical properties as θ is varied within this range?

As we can notice from the plots above, for $\theta \notin [-5.05, -2.9]$ the map as only one fixed point which is stable, while for $\theta \in [-5.05, -2.9]$ the map as three different fixed points, two unstable fixed points and one stable point in the middle.

3 Task 3 - Nonlinear systems, oscillations, and chaos

Consider the 'Ricker map':

$$x_{t+1} = r x_t e^{-x_t}, \quad r \in \mathbb{R},\ x_t \in \mathbb{R}$$

3.0.1 1) What are the fixed point(s) of this map? How many are there?

Analytical solution:

$$x^{\star} = r x^{\star} e^{-x^{\star}}$$

$$\begin{cases} x^{\star} = 0 & \forall r \\ \\ x^{\star} = log(r) & \forall r > 0 \end{cases}$$

Stability:

$$|F'(x_t)| = |re^{-x_t}(1 - x_t)| < 1$$

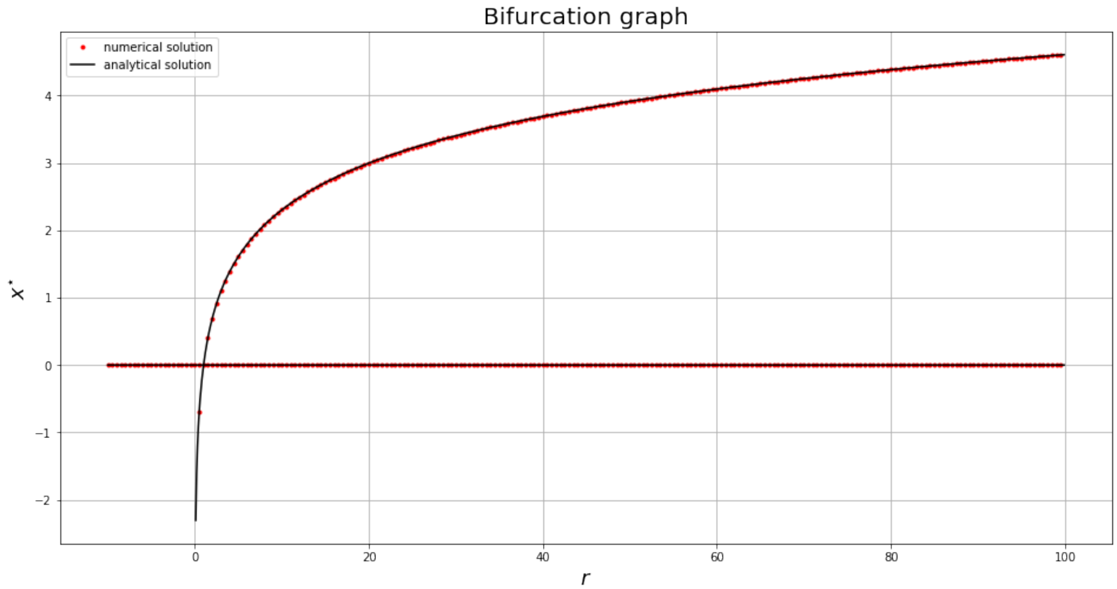
$$\begin{cases} F'(0) = r & stable\ for\ -1 < r < 1 \\ \\ F'(log(r)) = 1 - log(r) & stable\ \ for\ 1 < r < e^2 \end{cases}$$

In [16]:
```
# define Ricker map
def ricker(x, r):
    return r * x * np.exp(-x)
```

In [17]:
```
# compute fixed points numerically
r = np.arange(-10., 100, 0.5)

bifurcation = []

eps = 5e-5
x = np.arange(-10.,50, eps)

for i in r:
    fx = ricker(x, i)
    bifurcation.append( fixed_points(x,fx) )
```

In [18]:
```
# plot bifurcation graph
fig1 = plt.subplots(figsize=[16,8])

# plot numerical solution
plt.plot(np.zeros(bifurcation[0].shape)+r[0], bifurcation[0], '.r', markersize=6, label
for i in range(1,len(r)):
    plt.plot(np.zeros(bifurcation[i].shape)+r[i], bifurcation[i], '.r', markersize=6)

# plot analytical solution
r_an = np.arange(-10., 100., 0.1)
plt.plot(r_an, np.zeros(len(r_an)), 'k')
r_an = np.arange( 0.1, 100., 0.1)
plt.plot(r_an, np.log(r_an), 'k', label='analytical solution')

plt.xlabel(r'$r$', fontsize = 18)
plt.ylabel(r'$x^{\star}$', fontsize = 18)
```

```
plt.grid()
plt.legend()
plt.title('Bifurcation graph', fontsize = 20)
plt.show()
```

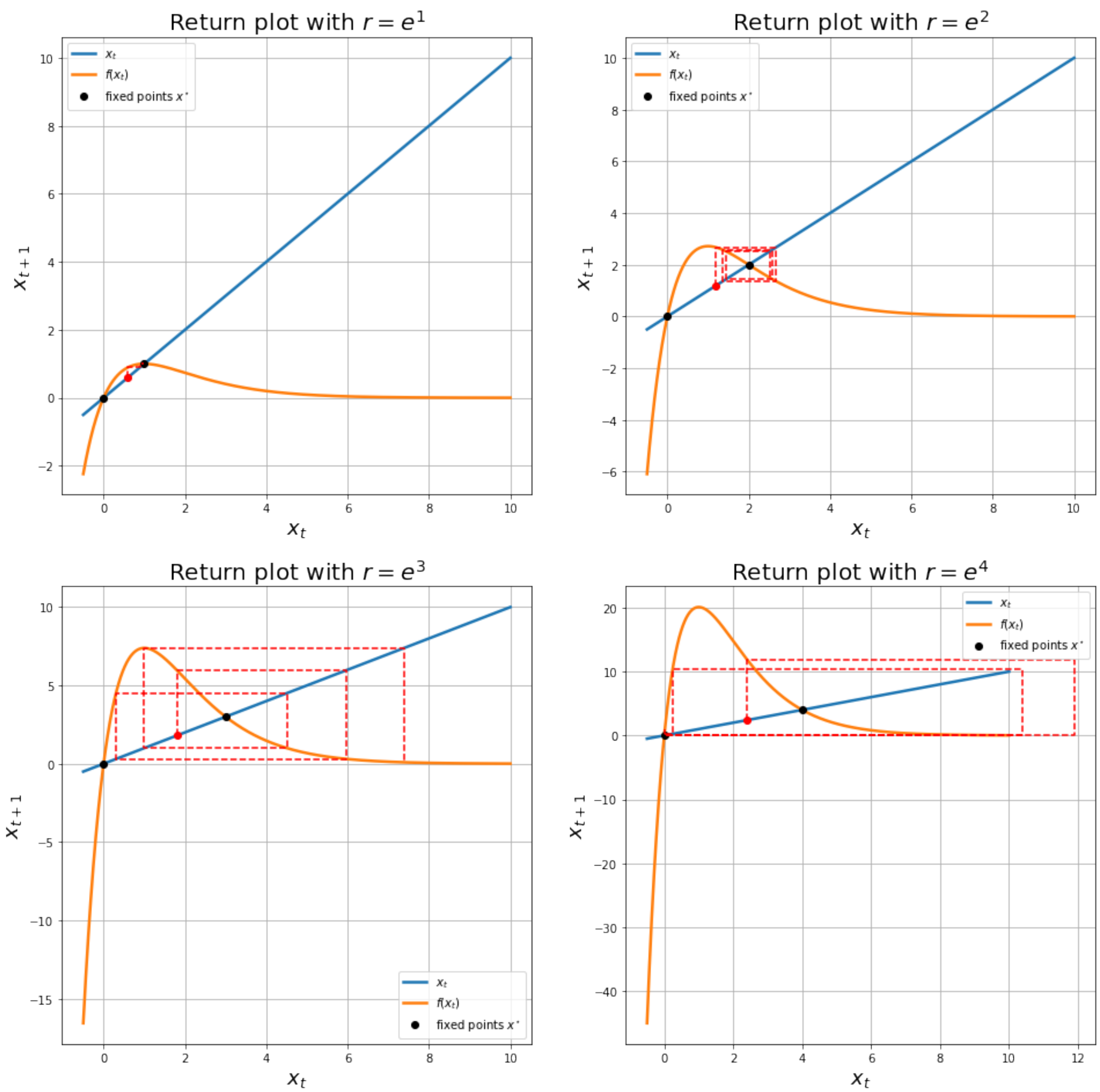


3.0.2 2) Explore the behavior of the map for a few values r ∈ [exp(1), . . . , exp(4)] (covering the extremes of this interval), and comment on the dynamics.

In [19]:
```
def cobweb(x0, r, steps = 12):
    p = np.zeros(( 2, steps ))

    p[:,0] = [x0, x0]
    p[:,1] = [x0, ricker(x0, r) ]

    for i in range(2, steps, 2):
        p[:,i] = [ p[1,i-1], p[1,i-1] ]
        p[:,i+1] = [ p[0,i], ricker(p[0,i], r) ]

    return p
```

In [20]:
```
# Compute fixed points for some values of parameter r as example
r = [1, 2, 3, 4]

x = np.arange(-0.5,10., eps)

# Plot the return plot
fig1 = plt.subplots(figsize=[16,16])
```

```python
for i in r:
    # compute function and fixed points
    fx = ricker(x, np.exp(i))
    fp = fixed_points(x, fx)

    # compute cobweb
    x1 = 3./5.*fp[1]
    x2 = 9./5.*fp[1]
    line1 = cobweb(x1, np.exp(i))
    #line2 = cobweb(x2, np.exp(i))

    # plot bifurcation graph
    plt.subplot(2,2,i)

    plt.plot(x, x, label=r'$x_t$', linewidth=2.5 )
    plt.plot(x, fx, label=r'$f(x_t)$', linewidth=2.5 )
    plt.plot(fp, fp, 'ok', label=r'fixed points $x^{\star}$', linewidth=3)
    plt.plot(line1[0], line1[1], '--r')
    plt.plot(line1[0,0], line1[1,0], 'or')
    #plt.plot(line2[0], line2[1], '--r')
    #plt.plot(line2[0,0], line2[1,0], 'or')
    plt.xlabel(r'$x_{t}$', fontsize = 18)
    plt.ylabel(r'$x_{t+1}$', fontsize = 18)
    plt.grid()
    plt.legend()
    plt.title(r'Return plot with $r=e^{%i}$' %(i) , fontsize = 20)

plt.show(fig1)
```



We have analytically demonstrated that in the range $r \in [e^1, e^4]$ the fixed point r=0 is instable and the plots above confirms that. In addition we have shown that the second fixed point is stable in the range $r \in [1, e^2]$: that is shown in the first plot by the cobweb plot, which is converging into the fixed point. On the other hand, in the other plots we can notice that the cobweb plot is diverging and this is a confirmation of the instable nature of the fixed points.