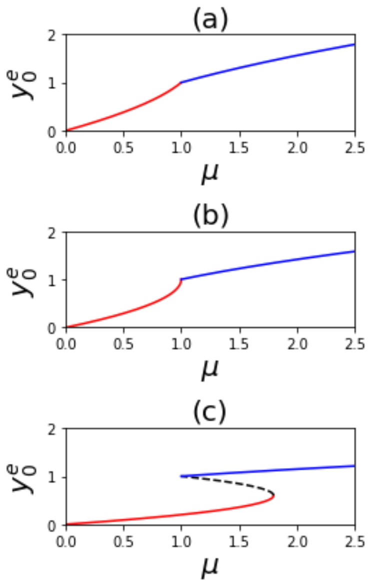
In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math
```

Fig. 6: 1D Bifurcation Diagram

In [2]:
```
plt.subplots(3, 1, sharey=True, figsize=(4, 6))
def f1(y):
    return y+lam*y*(1-y)
def f2(y):
    return y-lam*y*(1-y)
x = np.linspace(0, 1, 1000)
y = np.linspace(1, 2, 1000)

lam=0.5
plt.subplot(3, 1, 1)
plt.xlabel('$\mu$',fontsize=20)
plt.ylabel('$y_0^e$',fontsize=20)
plt.xlim(0,2.5)
plt.ylim(0,2)
plt.title('(a)', fontsize=20)
plt.plot(f1(x), x,'r');
plt.plot(f2(y), y, 'b');
lam=1
plt.subplot(3, 1, 2)
plt.xlabel('$\mu$',fontsize=20)
plt.ylabel('$y_0^e$',fontsize=20)
plt.xlim(0,2.5)
plt.ylim(0,2)
plt.title('(b)', fontsize=20)
plt.plot(f1(x), x,'r');
plt.plot(f2(y), y, 'b');
lam=5
plt.subplot(3, 1, 3)
x = np.linspace(0, 0.6, 1000)
y = np.linspace(0.6, 1, 1000)
z = np.linspace(1, 2, 1000)
plt.xlabel('$\mu$',fontsize=20)
plt.ylabel('$y_0^e$',fontsize=20)
plt.xlim(0,2.5)
plt.ylim(0,2)
plt.title('(c)', fontsize=20)
plt.plot(f1(x), x,'r');
plt.plot(f1(y), y,'black', linestyle='dashed');
plt.plot(f2(z), z, 'b');
plt.tight_layout()
```



Another method of plotting bifurcation diagram

Runge Kutta Method of Order Four

In [3]:
```
from mpl_toolkits.mplot3d import Axes3D
```

Define function

In [4]:
```
def f(x_value,y_value,mu_vaue):
    return 1.0/eps * (1 - x_value) - lam * abs(x_value - y_value) * x_value
def g(x_value,y_value,mu_value):
    return mu_value - y_value - lam * abs(x_value - y_value) * y_value
def h(x_value,y_value,mu_value,r):
    return r

def main(x_init,y_init,mu_init,delta_t,N):
    x = np.zeros(N)
    y = np.zeros(N)
    mu = np.zeros(N)
    x[0]=(x_init)
    y[0]=(y_init)
    mu[0]=(mu_init)
    for i in range(1,int(N)):
        k1 = f(x_init,y_init,mu_init) * delta_t
        k2 = f(x_init + delta_t/2.0,y_init + k1/2.0,mu_init) * delta_t
        k3 = f(x_init + delta_t/2.0,y_init + k2/2.0,mu_init) * delta_t
        k4 = f(x_init + delta_t,y_init + k3,mu_init) * delta_t
        x_iteration = x_init + 1.0/6.0 * (k1 + 2*k2 + 2*k3 + k4)
        l1 = g(x_init,y_init,mu_init) * delta_t
        l2 = g(x_init + delta_t/2.0,y_init + l1/2.0,mu_init) * delta_t
        l3 = g(x_init + delta_t/2.0,y_init + l2/2.0,mu_init) * delta_t
        l4 = g(x_init + delta_t,y_init + l3,mu_init) * delta_t
        y_iteration = y_init + 1.0/6.0 * (l1 + 2*l2 + 2*l3 + l4)
        m1 = h(x_init,y_init,mu_init,r) * delta_t
        m2 = h(x_init + delta_t/2.0,y_init + m1/2.0,mu_init,r) * delta_t
        m3 = h(x_init + delta_t/2.0,y_init + m2/2.0,mu_init,r) * delta_t
        m4 = h(x_init + delta_t,y_init + m3,mu_init,r) * delta_t
        mu_iteration = mu_init + 1.0/6.0 * (m1 + 2*m2 + 2*m3 + m4)

        x[i]=(x_iteration)
        y[i]=(y_iteration)
        mu[i]=(mu_iteration)
        x_init = x_iteration
        y_init = y_iteration
        mu_init = mu_iteration
    total_vec = [x,y,mu]
    return total_vec
```

In [5]:
```
eps = 0.01
lam = 5
```

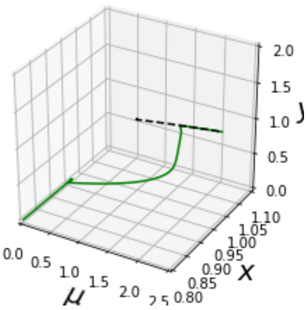
Plot for initial conditions (0,0,0) and (1,1,1.01). mu is increasing smoothly. Stepsize is too large since start of trajectory in first diagram goes too far.

In [6]:
```
from mpl_toolkits.mplot3d import Axes3D
```

In [7]:
```
r = 0.005
t = np.linspace(0,500,50000)
n = np.linspace(200,500,30000)
fig = plt.figure(figsize=(4, 4))
ax = plt.axes(projection='3d')
xline0 = main(0.8,0,0,0.01,t.size)[0]
xline1 = main(1,1,1.01,0.01,n.size)[0]
yline0 = main(0.8,0,0,0.01,t.size)[1]
yline1 = main(1,1,1.01,0.01,n.size)[1]
muline0 = main(0.8,0,0,0.01,t.size)[2]
muline1 = main(1,1,1.01,0.01,n.size)[2]

ax.plot3D(muline0, xline0, yline0, 'g')
ax.plot3D(muline1, xline1, yline1, 'black', linestyle="dashed")
ax.set_xlabel('$\mu$',fontsize=20)
ax.set_ylabel('$x$',fontsize=20)
ax.set_zlabel('$y$',fontsize=20)
ax.set_xlim(0, 2.5)
ax.set_ylim(0.8, 1.1)
ax.set_zlim(0, 2)
```

Out[7]: (0, 2)



Plot for initial conditions (1.2,2,2.5) and (1.2,0,1.7). mu is decreasing smoothly.

In [8]:
```
r = -0.005
t = np.linspace(0,500,50000)
n = np.linspace(160,500,34000)
fig = plt.figure(figsize=(4, 4))
ax = plt.axes(projection='3d')
xline0 = main(1,1.5,2.5,0.01,t.size)[0]
xline1 = main(1.1,0,1.7,0.01,n.size)[0]
yline0 = main(1,1.5,2.5,0.01,t.size)[1]
yline1 = main(1.1,0,1.7,0.01,n.size)[1]
muline0 = main(1,1.5,2.5,0.01,t.size)[2]
muline1 = main(1.1,0,1.7,0.01,n.size)[2]
ax.plot3D(muline0, xline0, yline0, 'g')
ax.plot3D(muline1, xline1, yline1, 'black', linestyle="dashed")
ax.set_xlabel('$\mu$',fontsize=20)
ax.set_ylabel('$x$',fontsize=20)
ax.set_zlabel('$y$',fontsize=20)
ax.set_xlim(0, 2.5)
ax.set_ylim(0.8, 1.1)
ax.set_zlim(0, 2)
```

Out[8]: (0, 2)

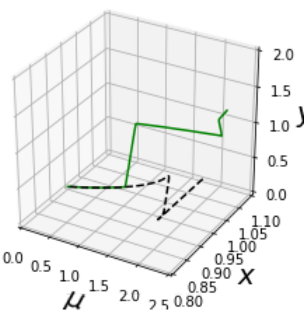


Figure 1:

In [9]:
```
fig = plt.figure(figsize=(4, 4))
ax = plt.axes(projection='3d')
t = np.linspace(0,3000,300000)
r = 0.005
xline0 = main(0.8,0,1.0,0.001,t.size)[0]
yline0 = main(0.8,0,1.0,0.001,t.size)[1]
muline0 = main(0.8,0,1.0,0.001,t.size)[2]
ax.plot3D(muline0, xline0, yline0, 'g')

n = np.linspace(0,2990,299000)
r = -0.005
xline1 = main(1.05,1.3,2.5,0.001,n.size)[0]
yline1 = main(1.05,1.3,2.5,0.001,n.size)[1]
muline1 = main(1.05,1.3,2.5,0.001,n.size)[2]
ax.plot3D(muline1, xline1, yline1, 'black', linestyle="dashed")
ax.set_xlabel('$\mu$',fontsize=20)
ax.set_ylabel('$x$',fontsize=20)
ax.set_zlabel('$y$',fontsize=20)
ax.set_xlim(1., 2.5)
ax.set_ylim(0.8, 1.1)
ax.set_zlim(0, 1.5)
```

Out[9]: (0, 1.5)

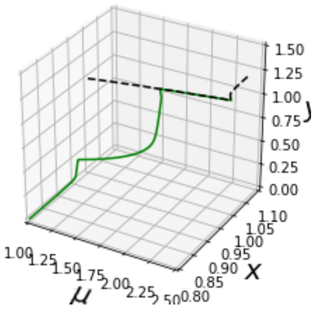


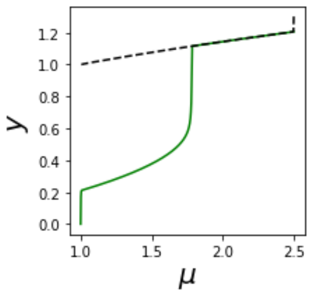
Fig. 11: Plot diagram I wish to put bifurcation diagram over on draw.io.

In [10]:
```
fig = plt.figure(figsize=(3, 3))

t = np.linspace(0,300,30000)
r = 0.005
xline0 = main(0.8,0,1.0,0.01,t.size)[0]
yline0 = main(0.8,0,1.0,0.01,t.size)[1]
muline0 = main(0.8,0,1.0,0.01,t.size)[2]
plt.plot(muline0, yline0, 'g')
n = np.linspace(0,299,29900)

r = -0.005
xline1 = main(1.05,1.3,2.5,0.01,n.size)[0]
yline1 = main(1.05,1.3,2.5,0.01,n.size)[1]
muline1 = main(1.05,1.3,2.5,0.01,n.size)[2]
plt.plot(muline1, yline1, 'black', linestyle="dashed")
plt.xlabel('$\mu$',fontsize=20)
plt.ylabel('$y$',fontsize=20)
```

Out[10]: Text(0,0.5,'y')



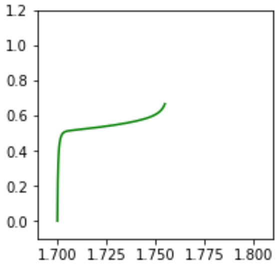
Return from Tipping

Want to test what value is acceptable for mu_b for Fig. 12

In [11]:
```python
fig = plt.figure(figsize=(3, 3))
t = np.linspace(0,1000,100000)  # careful of linspace (0,5000,500000) gives interval 0.5 width
r = 0.001
xline0 = main(0.8,0,1.7,0.00055,t.size)[0] # 0.005 gives interval 0.5 width
yline0 = main(0.8,0,1.7,0.00055,t.size)[1]
muline0 = main(0.8,0,1.7,0.00055,t.size)[2]
plt.plot(muline0, yline0, 'g')
plt.ylim(-0.1, 1.2)
plt.xlim(1.69, 1.81)

# linspace(_,_,X) * r * stepsize = width of evaluation space  =>  100000 * 0.001 * 0.001 = 100 * 0.001 = 0.1
```

Out[11]:
```
(1.69, 1.81)
```



Pick 1.755 as mu_b

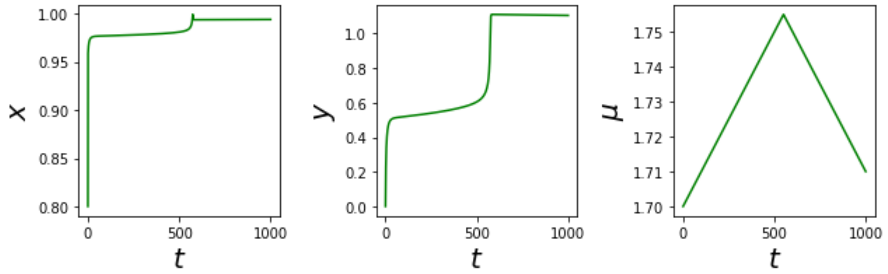
Creating a new Runge Kutta function which reverses the parameter mu at the value for mu_b we have picked.

In [12]:
```
def f(x_value,y_value,mu_vaue):
    return 1.0/eps * (1 - x_value) - lam * abs(x_value - y_value) * x_value
def g(x_value,y_value,mu_value):
    return mu_value - y_value - lam * abs(x_value - y_value) * y_value
def returntipping(x_init,y_init,mu_init,delta_t,N,r):
    x = np.zeros(N)
    y = np.zeros(N)
    mu = np.zeros(N)
    x[0]=(x_init)
    y[0]=(y_init)
    mu[0]=(mu_init)
    j=1
    for i in range(1,int(N)):
        if mu_init < 1.755:
            k1 = f(x_init,y_init,mu_init) * delta_t
            k2 = f(x_init + delta_t/2.0,y_init + k1/2.0,mu_init) * delta_t
            k3 = f(x_init + delta_t/2.0,y_init + k2/2.0,mu_init) * delta_t
            k4 = f(x_init + delta_t,y_init + k3,mu_init) * delta_t
            x_iteration = x_init + 1.0/6.0 * (k1 + 2*k2 + 2*k3 + k4)
            l1 = g(x_init,y_init,mu_init) * delta_t
            l2 = g(x_init + delta_t/2.0,y_init + l1/2.0,mu_init) * delta_t
            l3 = g(x_init + delta_t/2.0,y_init + l2/2.0,mu_init) * delta_t
            l4 = g(x_init + delta_t,y_init + l3,mu_init) * delta_t
            y_iteration = y_init + 1.0/6.0 * (l1 + 2*l2 + 2*l3 + l4)
            mu_iteration = mu_init + r * delta_t
            x[i]=(x_iteration)
            y[i]=(y_iteration)
            mu[i]=(mu_iteration)
            x_init = x_iteration
            y_init = y_iteration
            mu_init = mu_iteration
            j=i
        else:
            break
    r=-r
    for i in range(j,int(N)):
        k1 = f(x_init,y_init,mu_init) * delta_t
        k2 = f(x_init + delta_t/2.0,y_init + k1/2.0,mu_init) * delta_t
        k3 = f(x_init + delta_t/2.0,y_init + k2/2.0,mu_init) * delta_t
        k4 = f(x_init + delta_t,y_init + k3,mu_init) * delta_t
        x_iteration = x_init + 1.0/6.0 * (k1 + 2*k2 + 2*k3 + k4)
        l1 = g(x_init,y_init,mu_init) * delta_t
        l2 = g(x_init + delta_t/2.0,y_init + l1/2.0,mu_init) * delta_t
        l3 = g(x_init + delta_t/2.0,y_init + l2/2.0,mu_init) * delta_t
        l4 = g(x_init + delta_t,y_init + l3,mu_init) * delta_t
        y_iteration = y_init + 1.0/6.0 * (l1 + 2*l2 + 2*l3 + l4)
        mu_iteration = mu_init + r * delta_t
        x[i]=(x_iteration)
        y[i]=(y_iteration)
        mu[i]=(mu_iteration)
        x_init = x_iteration
        y_init = y_iteration
        mu_init = mu_iteration

    total_vec = [x,y,mu]
    return total_vec
```

Shows where mu is reversed and how x and y behave.

In [13]:
```
t = np.linspace(0,1000,100000)
plt.subplots(1, 3, sharey=True, figsize=(9, 3))
plt.subplot(1, 3, 1)
plt.plot(t,returntipping(0.8,0,1.7,0.001,t.size,0.001)[0],'g')
plt.xlabel("$t$",fontsize=20)
plt.ylabel("$x$",fontsize=20)
plt.subplot(1, 3, 2)
r=0.001
plt.plot(t,returntipping(0.8,0,1.7,0.001,t.size,0.001)[1],'g')
plt.xlabel("$t$",fontsize=20)
plt.ylabel("$y$",fontsize=20)
plt.subplot(1, 3, 3)
r=0.001
plt.plot(t,returntipping(0.8,0,1.7,0.001,t.size,0.001)[2],'g')
plt.xlabel("$t$",fontsize=20)
plt.ylabel("$\mu$",fontsize=20)
plt.tight_layout()
```



Check that parameter-shift reversal works. See if it tips for rate 0.0007

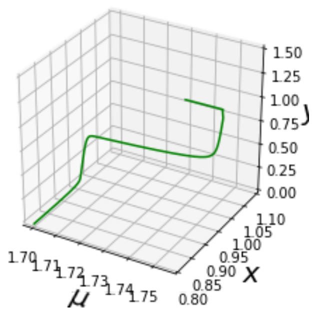
In [14]:
```
fig = plt.figure(figsize=(4, 4))
ax = plt.axes(projection='3d')

t = np.linspace(0,1000,100000)

xline0 = returntipping(0.8,0,1.7,0.001,t.size,0.0007)[0]
yline0 = returntipping(0.8,0,1.7,0.001,t.size,0.0007)[1]
muline0 = returntipping(0.8,0,1.7,0.001,t.size,0.0007)[2]
ax.plot3D(muline0, xline0, yline0, 'g')

ax.set_xlabel('$\mu$',fontsize=20)
ax.set_ylabel('$x$',fontsize=20)
ax.set_zlabel('$y$',fontsize=20)
ax.set_ylim(0.8, 1.1)
ax.set_zlim(0, 1.5)
```

Out[14]: (0, 1.5)



Check that system tips for r = 0.004

In [15]:
```
fig = plt.figure(figsize=(4, 4))
ax = plt.axes(projection='3d')
t = np.linspace(0,1000,20000)
xline0 = returntipping(0.8,0,1.7,0.001,t.size,0.004)[0]
yline0 = returntipping(0.8,0,1.7,0.001,t.size,0.004)[1]
muline0 = returntipping(0.8,0,1.7,0.001,t.size,0.004)[2]
ax.plot3D(muline0, xline0, yline0, 'g')
ax.set_xlabel('$\mu$',fontsize=20)
ax.set_ylabel('$x$',fontsize=20)
ax.set_zlabel('$y$',fontsize=20)
ax.set_ylim(0.8, 1.1)
ax.set_zlim(0, 1.5)
```

Out[15]: `(0, 1.5)`

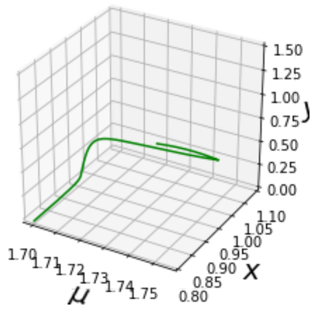


Fig. 12: Plot of trajectory (a) tipping, (b) following unstable solution and (c) not tipping.

In [16]:
```
plt.subplots(1, 3, sharey=True, figsize=(9, 3))
plt.subplot(1, 3, 1)
m=0.7
t = np.linspace(0,1000,105000.0/m)  # careful of linspace
xline0 = returntipping(0.8,0,1.7,0.001,t.size,0.001*m)[0]
yline0 = returntipping(0.8,0,1.7,0.001,t.size,0.001*m)[1]
muline0 = returntipping(0.8,0,1.7,0.001,t.size,0.001*m)[2]
plt.plot(muline0, yline0, 'g')
plt.xlabel('$\mu$',fontsize=20)
plt.ylabel('$y$',fontsize=20)
plt.xlim(1.68, 1.77)
plt.ylim(0.3, 1.2)
plt.subplot(1, 3, 2)

m=2.45889
t = np.linspace(0,1000,105000.0/m)
xline0 = returntipping(0.8,0,1.7,0.001,t.size,0.001*m)[0]
yline0 = returntipping(0.8,0,1.7,0.001,t.size,0.001*m)[1]
muline0 = returntipping(0.8,0,1.7,0.001,t.size,0.001*m)[2]
plt.plot(muline0, yline0, 'g')
plt.xlabel('$\mu$',fontsize=20)
plt.ylabel('$y$',fontsize=20)
plt.xlim(1.68, 1.77)
plt.ylim(0.3, 1.2)
plt.subplot(1, 3, 3)

m=4.0
t = np.linspace(0,1000,105000.0/m)
xline0 = returntipping(0.8,0,1.7,0.001,t.size,0.001*m)[0]
yline0 = returntipping(0.8,0,1.7,0.001,t.size,0.001*m)[1]
muline0 = returntipping(0.8,0,1.7,0.001,t.size,0.001*m)[2]
plt.plot(muline0, yline0, 'g')
plt.xlabel('$\mu$',fontsize=20)
plt.ylabel('$y$',fontsize=20)
plt.xlim(1.68, 1.77)
plt.ylim(0.3, 1.2)
```

```
C:\Users\Laura\Anaconda2\lib\site-packages\ipykernel_launcher.py:4: DeprecationWarning: object of type <type 'floa
t'> cannot be safely interpreted as an integer.
  after removing the cwd from sys.path.
C:\Users\Laura\Anaconda2\lib\site-packages\ipykernel_launcher.py:16: DeprecationWarning: object of type <type 'flo
at'> cannot be safely interpreted as an integer.
  app.launch_new_instance()
C:\Users\Laura\Anaconda2\lib\site-packages\ipykernel_launcher.py:28: DeprecationWarning: object of type <type 'flo
at'> cannot be safely interpreted as an integer.
```

Out[16]: (0.3, 1.2)