HW01

Andy Reagan

January 26, 2014

1 problem 1

Consider the IVP

$$y' = ay, y(x_0) = y_0,$$

where a is a constant. Following the lines of derivation of the global error, find how the sign of the global error depends on y_0 and a.

1.1 solution

from equation 1.4 in the notes, we have the following expression for the global error:

$$\epsilon_{i+1} = \epsilon_i + h(f(x_i, y_i) - f(x_i, Y_i)) + \frac{h^2}{2} y''(x_i^*).$$

for our equation, we have $y(x) = y_0 e^{a(x-x_0)}$ and equation 1.4 becomes

$$\epsilon_{i+1} = \epsilon_i + ha(y_i - Y_i) + \frac{h^2}{2} a^2 y_0 e^{a(x_i^* - x_0)}.$$

noticing that by defition, $\epsilon_i = y_i - Y_i$, we can rewrite the above as

$$\epsilon_{i+1} = (1 + ah)\epsilon_i + \frac{h^2}{2} a^2 y_0 e^{a(x_i^* - x_0)}.$$

leaving out the terms that are always positive, we then have (assuming $1 + ah > 0$, and denoting sgn the sign function)

$$sgn(\epsilon_{i+1}) = sgn(\epsilon_i + y_0).$$

now since ϵ_0 is the local error (the last term in 1.4), the above has shown that the sign of ϵ_0 is the same as the sign of y_0. therefore, the sign of the global error follows the sign of y_0.

In [2]:
```
from numpy import *

def sign(x):
    if x<0:
        return 'negative'
    else:
        return 'positive'

h = 0.1
x = linspace(0,1,num=int(1/h+1))

for a in [1,-1]:
    for y0 in [1,-1]:
        y = y0
        for i in xrange(10):
            y += h*a*y

        print 'for y0={0},a={1}, the sign of global error is {2}'.format(y0,a,sign(y))
```

```
for y0=1,a=1, the sign of global error is positive
for y0=-1,a=1, the sign of global error is negative
for y0=1,a=-1, the sign of global error is positive
for y0=-1,a=-1, the sign of global error is negative
```

2 problem 2

find the coefficient B in the midpoint method (1.19)

2.1 solution

our scheme is written as

$$Y_{i+1} = Y_i + hf(x_i + h/2, Y_i + Bh).$$

to find B, we compare the taylor expansion of this scheme and the analytical solution. the taylor series for the numerical scheme is thus

$$Y_{i+1} = Y_i + hf(x_i, Y_i) + \frac{h^2}{2} f_x(x_i, Y_i) + Bh^2 f_y(x_i, Y_i) + O(h^3).$$

from the notes, the taylor series for the analytical solution is

$$y_{i+1} = y_i + hf(x_i, y_i) + \frac{h^2}{2}(f_x(x_i, Y_i) + f_y(x_i, Y_i) f(x_i, y_i)) + O(h^3).$$

to compute the local error, assume that $y_i = Y_i$ and subtracting these two equations above, we have

$$\epsilon_{i+1} = \frac{h^2}{2} f_y(x_i, Y_i) f(x_i, y_i) - Bh^2 f_y(x_i, Y_i) + O(h^3).$$

setting $\epsilon_{i+1} = O(h^3)$ and solving for B we have that

$$B = f(x_i, y_i)/2.$$

3 problem 3

Derive the expression for the local truncation error of the Midpoint method.

3.1 solution

first, I adopt the notation used in the notes, and write the analytical taylor series as

$$y_{i+1} = y_i + hf + \frac{h^2}{2}(f_x + f f_y) + \frac{h^3}{6}(f_{xx} + f_x f_y + 2f f_{xy} + f(f_y)^2 + f^2 f_{yy}) + O(h^4).$$

now for the midpoint method, we have

$$Y_{i+1} = Y_i + hf + \frac{h^2}{2}(f_x + ff_y) + \frac{h^3}{8}(f_{xx} + 2ff_{xy} + f^2 f_{yy}) + O(h^4).$$

subtracting these two, the error is

$$\epsilon_{i+1} = h^3 \left[\left(\frac{1}{6} - \frac{1}{8} \right) (f_{xx} + 2ff_{xy} + f^2 f_{yy}) + \frac{1}{6} \left(f_x f_y + f(f_y)^2 \right) \right] + O(h^4)$$

and simplifying, this is

$$\epsilon_{i+1} = h^3 \left[\frac{1}{24} (f_{xx} + 2ff_{xy} + f^2 f_{yy}) + \frac{1}{6} \left(f_x f_y + f(f_y)^2 \right) \right] + O(h^4).$$

4 problem 4

Solve the IVP $y' = \sqrt{y}, y(0) = 1$ for $x \in [0, 1]$ using three methods: simple Euler, modified Euler, and Midpoint. In each case, use two values for the step size: $h = 0.1$ and $h = 0.05$. Make a table that shows your numerical error. By what factor does the error decrease when the step size decreases from 0.1 to 0.05?

Are you results consistent with the expressions for the the local truncation errors of these three methods? Namely:

1. Do the dependencies of theses errors on h agree with the theoretical predictions?
2. Does the sign of the error from the simple Euler method agree with that which follows from the derivation in Section 1.3 for this specific $y(x)$?
3. Do the signs and relative magnitudes of the errors of the modified Euler and Midpoint methods agree with the result of Sec 1.6 and your result in problem 3?

4.1 solution

In [11]:
```
truth = ((1.0+2.0)**2)/4.0

y0 = 1.0

print
print '-'*52
print '     h          euler        mod euler     midpoint  \n'
print '-'*52

# loop over step size
for h in [.05,.1]:
    x = linspace(0,1,num=int(1/h+1))

    print '{0:8.03f} '.format(h),

    # simple euler
    y = y0
    for i in x[1:]:
        y = y+h*sqrt(y)

    print'  {0:8.03f}    '.format(truth-y),

    # modified euler
    y = y0
    for i in x[1:]:
        k1 = sqrt(y)
        k2 = sqrt(y+h*k1)
        y = y + 0.5*h*(k1+k2)
    print '    {0:8.05f}   '.format(truth-y),

    # midpoint
    y = y0
    for i in x[1:]:
        k1 = sqrt(y)
        k2 = sqrt(y+0.5*h*k1)
        # sqrt(y+h*0.5*sqrt(y))
        y = y+h*k2
    print'    {0:8.05f}  \n'.format(truth-y)
    print '-'*52
```

```
----------------------------------------------------
     h          euler        mod euler     midpoint

----------------------------------------------------
   0.050        0.015         0.00015        0.00008

----------------------------------------------------
   0.100        0.030         0.00060        0.00031

----------------------------------------------------
```

The errors decrease linearly for the Euler method (a factor of 2) and by a factor of 4 for the second-order methods. This is expected from the accuracy of these schemes. The dependencies on h do agree with the theory.

The sign of the error for Simple Euler does agree with section 1.3, since $y_0 > 0$ and $y'' > 0$ would predict an error $>$ 0. Indeed the error is greater than 0.

Both errors are positive, and this partially agrees with prediction, since $y_0 > 0$ and $y'' > 0$. The Midpoint method is approximately twice as accurate here, and this relative magnitude of accurarcy increase agrees. Our theory has the coefficient of -1/12 in the error of the ME method, and the coefficient of 1/24 in the Midpoint method, so we would expect this halving of the error seen here.

The term with leading 1/6 in both errors is 1/24 for this y, and the other term remains with $-1/4 \cdot \sqrt{y}$. What does not agree is the sign of the Midpoint error, which we would now expect to be negative, but is positive here.

5 problem 5

A skydiver jumps into a fluid that is resistant with direct proportion to his velocity v.

5.1 solution

Before his parachute opens to slow him, his velocity follows the ODE:

$$\frac{dv}{dt} = g - \frac{k}{m}v.$$

Setting the parameters such that his maximum velocity reaches 80 m/s, we solve for $dv/dt = 0$ for his asymptotical terminal velocity. This is solving

$$0 = 9.8 - 80\frac{k}{m}.$$

Since there are two unknowns, and one equation, we do not consider mass. This results in k equal to

```
In [8]:
```

```
g = 9.8
k = g/35.76 # that's 80mph in m/s
print k
```

```
0.274049217002
```

and a new ODE

$$\frac{dv}{dt} = g - kv.$$

The analytical solution, solved by separating the ODE into

$$\frac{dv}{g - kv} = dt$$

and integrating to obtain

$$-\ln(g - kv)/k = t + C$$

and solving for v as

$$v(t) = \frac{-1}{k}e^{-k(t+C)} + \frac{g}{k}.$$

Using the IC we set $C = -\ln(g)/k$. This yields the ODE

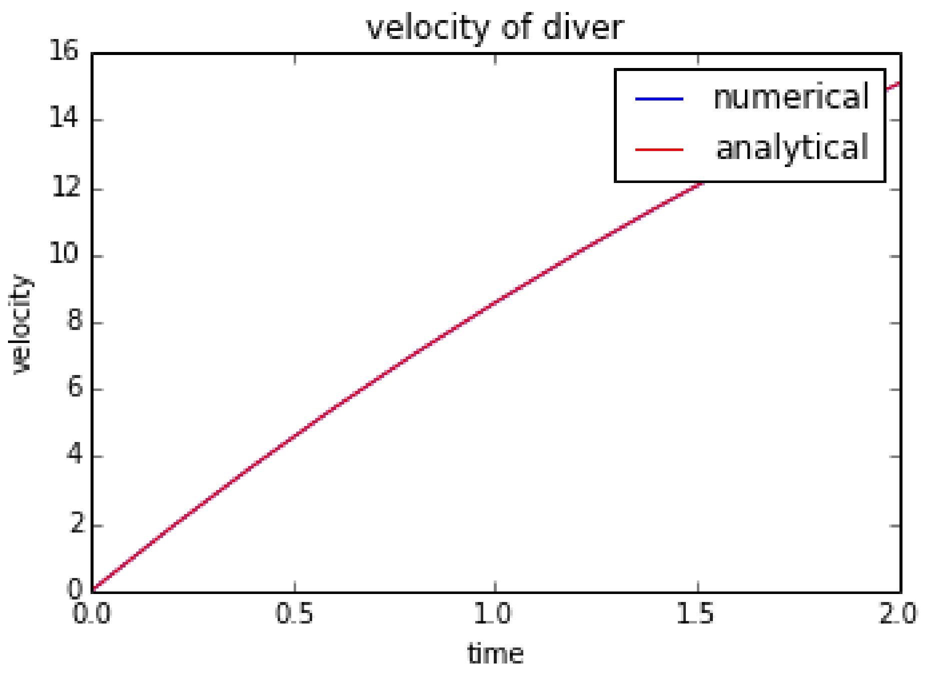
$$v(t) = \frac{g}{k}\left(-e^{-kt} + 1\right).$$

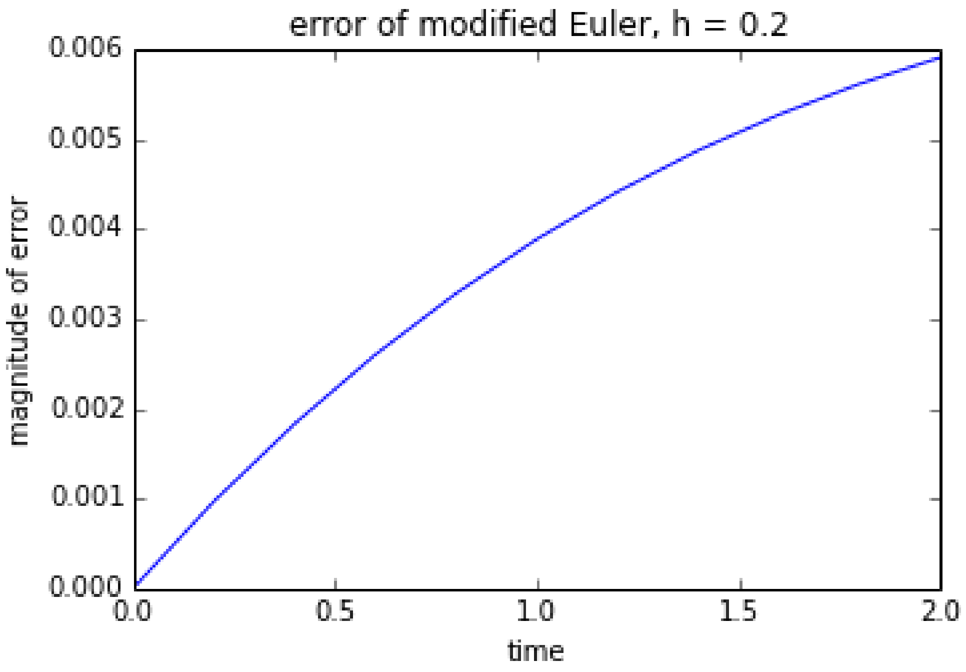
In [9]:
```
h = 0.2
c = -log(g)/k
y0 = 0.0
x = linspace(0,2,num=int(2/h+1))
print x
# initialize the solutions as vectors
yNum = linspace(0,2,num=int(2/h+1))
yAnal = linspace(0,2,num=int(2/h+1))

def jumperV(t,v,g,k):
    return g-v*k

# start at 0
yNum[0] = y0
yAnal[0] = y0
for t in xrange(1,len(x)):
    k1 = jumperV(t,yNum[t-1],g,k)
    k2 = jumperV(t+h,yNum[t-1]+h*k1,g,k)
    yNum[t] = yNum[t-1]+h/2*(k1+k2)
    yAnal[t] = -g/k*(exp(-k*x[t])-1)
print yNum
print yAnal
```

```
[ 0.   0.2  0.4  0.6  0.8  1.   1.2  1.4  1.6  1.8  2. ]
[  0.           1.90628635   3.71095281   5.41941649   7.03680576
   8.56797559  10.01752215  11.3897966   12.68891814  13.91878638
  15.08309308]
[  0.           1.9072544    3.71278566   5.42201918   7.04009097
   8.57186314  10.02193846  11.39467423  12.69419534  13.92440667
  15.08900487]
```

In [10]:
```
%matplotlib inline
import matplotlib.pyplot as plt
fig = plt.figure()
ax1 = fig.add_axes([0.15,0.2,0.7,0.7]) #  [left, bottom, width, height]
```

```
ax1.plot(x,yNum,'b')
ax1.plot(x,yAnal,'r')
ax1.legend(['numerical','analytical'])
plt.title('velocity of diver')
plt.xlabel('time')
plt.ylabel('velocity')
plt.show()

plt.close(fig)

fig = plt.figure()
ax1 = fig.add_axes([0.15,0.2,0.7,0.7]) #  [left, bottom, width, height]

ax1.plot(x,abs(yAnal-yNum))
plt.title('error of modified Euler, h = {0:g}'.format(h))
plt.xlabel('time')
plt.ylabel('magnitude of error')
plt.show()

plt.close(fig)
```




error of modified Euler, h = 0.2
magnitude of error
time
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.0
0.5
1.0
1.5
2.0