Lab-2

October 13, 2016

1 Lab 2

```
In [1]: %matplotlib inline
        import numpy
        from matplotlib import pyplot
        import scipy.integrate
        import scipy.optimize

In [2]: v = (1.0 + 1e-4 * numpy.random.rand(8))
        print(v)
```

```
[ 1.00006614  1.00006701  1.00004293  1.00005639  1.00005695  1.00005809
  1.0000406   1.00007049]
```

```
In [3]: A = numpy.diagflat(v)
        B = numpy.diagflat(v, -1)
        print(A)
        print(B)
```

```
[[ 1.00006614  0.          0.          0.          0.          0.          0.
   0.        ]
 [ 0.          1.00006701  0.          0.          0.          0.          0.
   0.        ]
 [ 0.          0.          1.00004293  0.          0.          0.          0.
   0.        ]
 [ 0.          0.          0.          1.00005639  0.          0.          0.
   0.        ]
 [ 0.          0.          0.          0.          1.00005695  0.          0.
   0.        ]
 [ 0.          0.          0.          0.          0.          1.00005809
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   1.0000406   0.        ]
 [ 0.          0.          0.          0.          0.          0.          0.
   1.00007049]]
[[ 0.          0.          0.          0.          0.          0.          0.
   0.          0.        ]
```

```
[ 1.00006614  0.          0.          0.          0.          0.          0.
  0.          0.        ]
[ 0.          1.00006701  0.          0.          0.          0.          0.
  0.          0.        ]
[ 0.          0.          1.00004293  0.          0.          0.          0.
  0.          0.        ]
[ 0.          0.          0.          1.00005639  0.          0.          0.
  0.          0.        ]
[ 0.          0.          0.          0.          1.00005695  0.          0.
  0.          0.        ]
[ 0.          0.          0.          0.          0.          1.00005809
  0.          0.          0.        ]
[ 0.          0.          0.          0.          0.          0.
  1.0000406   0.          0.        ]
[ 0.          0.          0.          0.          0.          0.          0.
  1.00007049  0.        ]]
```

```
In [4]: A1 = numpy.reshape(v, (2, 4))
        A2 = numpy.reshape(v, (4, 2))
        A3 = numpy.reshape(v, (2, 2, 2))
        print(A1)
        print(A2)
        print(A3)
```

```
[[ 1.00006614  1.00006701  1.00004293  1.00005639]
 [ 1.00005695  1.00005809  1.0000406   1.00007049]]
[[ 1.00006614  1.00006701]
 [ 1.00004293  1.00005639]
 [ 1.00005695  1.00005809]
 [ 1.0000406   1.00007049]]
[[[ 1.00006614  1.00006701]
  [ 1.00004293  1.00005639]]

 [[ 1.00005695  1.00005809]
  [ 1.0000406   1.00007049]]]
```

1.1 Quadrature

```
In [5]: def f1(x):
            return numpy.sin(x)**2
```

We have already imported all the libraries at the top, so we don't need to do it here. If we'd defined `f1` in a separate file we would need to import `numpy` into that file, and we'd then need to import `f1` into this one to use it.

```
In [6]: q = scipy.integrate.quad(f1, 0, numpy.pi)
        print(q)
```

```
(1.5707963267948966, 1.743934249004316e-14)
```

This returns both the result, but also the estimated error in the computation of the result.

```
In [7]: f2 = lambda x: numpy.sin(x)**2
        q2 = scipy.integrate.quad(f2, 0, numpy.pi)
        print(q2)
```

```
(1.5707963267948966, 1.743934249004316e-14)
```

```
In [8]: def f3(x):
            return numpy.exp(-x**2 * numpy.cos(2.0 * numpy.pi * x)**2)

In [9]: q3 = scipy.integrate.quad(f3, 0, 1)
        print(q3)
```

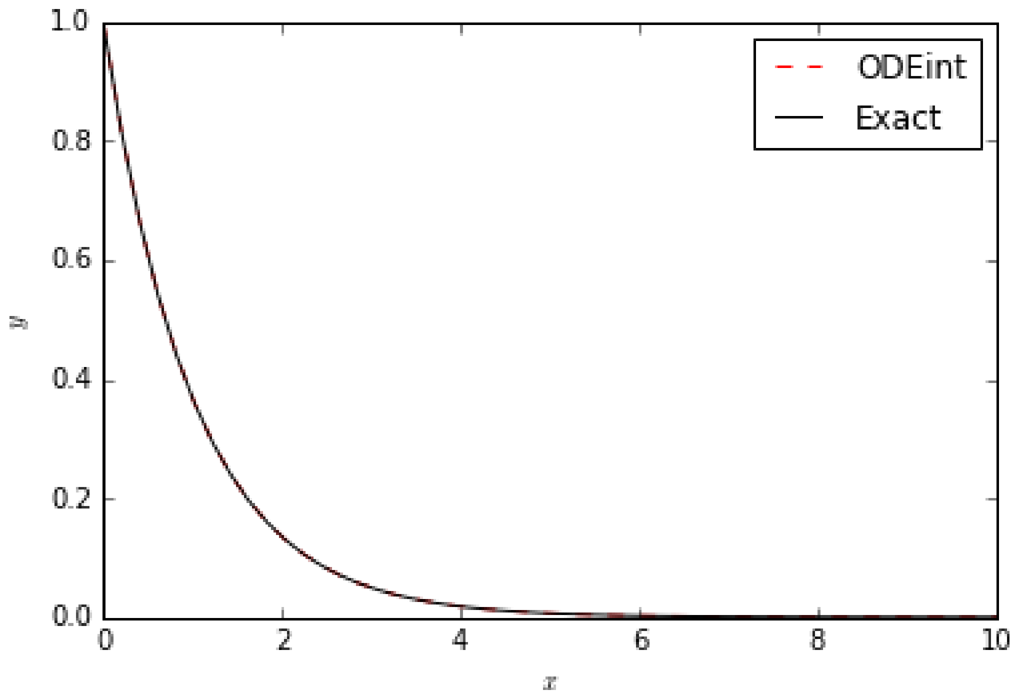
```
(0.8624119289380107, 9.617331941225508e-11)
```

```
In [10]: q3_accurate = scipy.integrate.quad(f3, 0, 1, epsabs = 1e-14, epsrel = 1e-1
         print(q3_accurate)
```

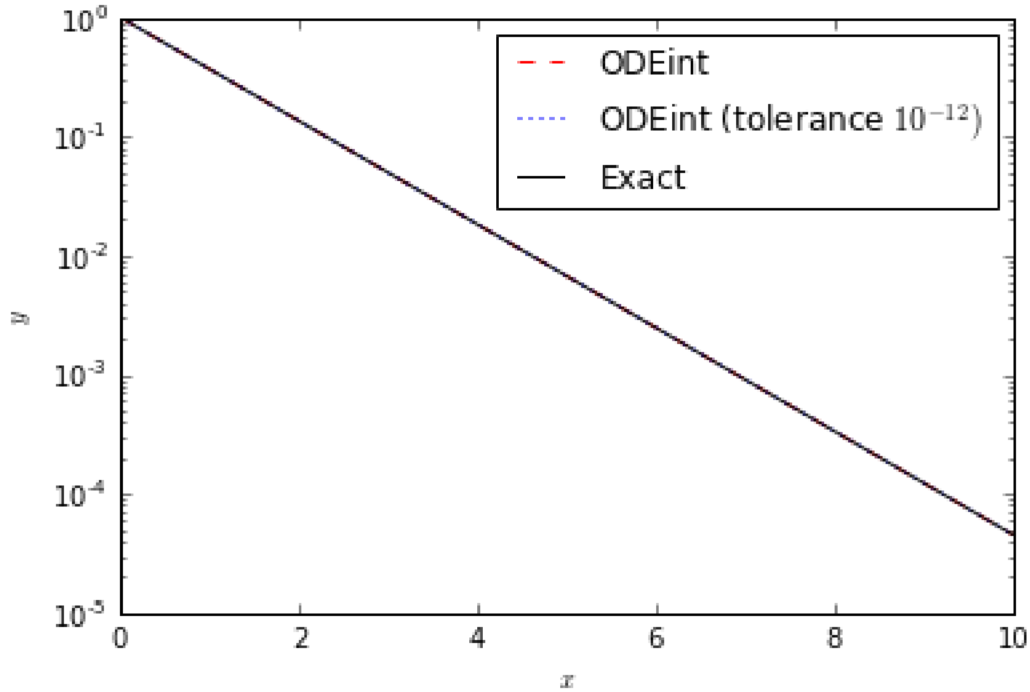
```
(0.8624119289380108, 9.574714734243736e-15)
```

1.2 Ordinary Differential Equations

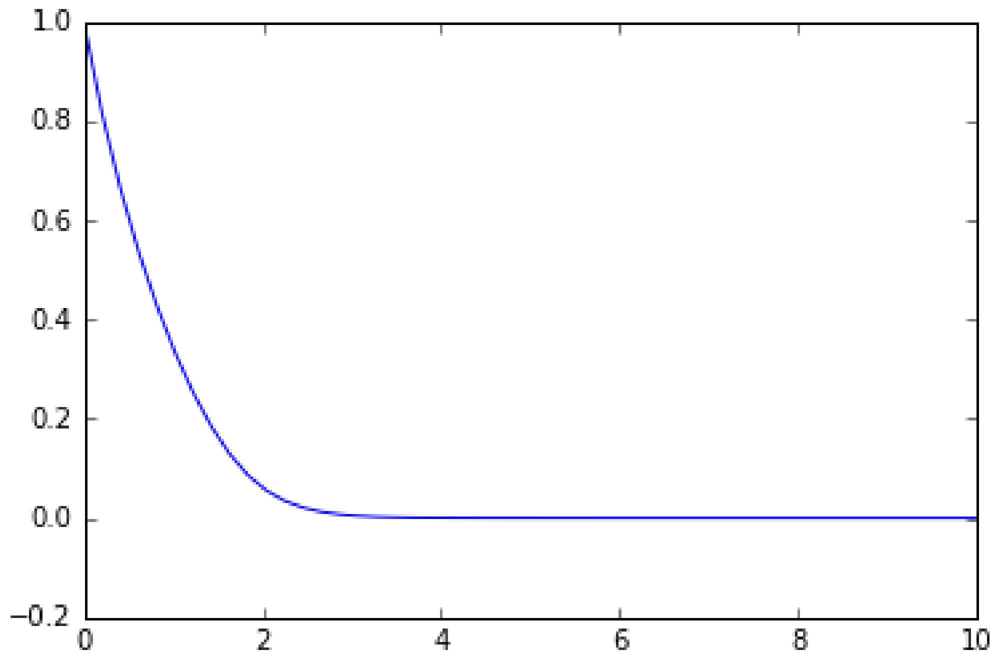
```
In [11]: f = lambda y, x: -y
         x = numpy.linspace(0, 10)
         solivp = scipy.integrate.odeint(f, [1.0], x)

In [12]: pyplot.plot(x, solivp, 'r--', label="ODEint")
         pyplot.plot(x, numpy.exp(-x), 'k-', label="Exact")
         pyplot.xlabel(r"$x$")
         pyplot.ylabel(r"$y$")
         pyplot.legend();
```




```
In [13]: solivp_accurate = scipy.integrate.odeint(f, [1.0], x, rtol = 1e-12, atol =

In [14]: pyplot.semilogy(x, solivp, 'r--', label="ODEint")
         pyplot.semilogy(x, solivp_accurate, 'b:', label=r"ODEint (tolerance $10^{-
         pyplot.semilogy(x, numpy.exp(-x), 'k-', label="Exact")
         pyplot.xlabel(r"$x$")
         pyplot.ylabel(r"$y$")
         pyplot.legend();
```



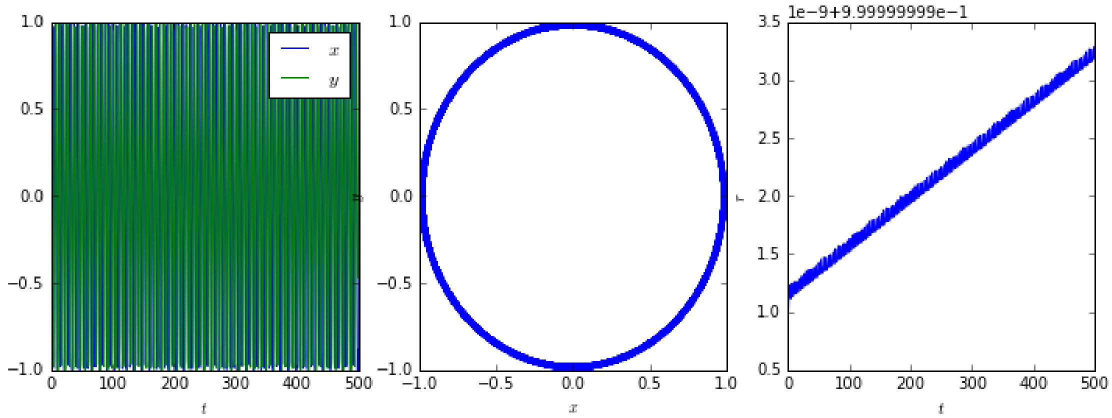

```
In [15]: def integrand(s):
             return numpy.sin(s)**2

         def C(x):
             return 1.0 + scipy.integrate.quad(integrand, 0, x)[0]

         def g(y, x):
             return -C(x) * y

         solivp = scipy.integrate.odeint(g, [1.0], x)

In [16]: pyplot.plot(x, solivp)

Out[16]: [<matplotlib.lines.Line2D at 0x10f417e80>]
```




```
In [17]: def system(z, t):
             dzdt = numpy.zeros_like(z)
             x = z[0]
             y = z[1]

             dzdt[0] = -y
             dzdt[1] = x

             return dzdt

         z0 = [1.0, 0.0]
         t = numpy.linspace(0, 500, 1000)

         z = scipy.integrate.odeint(system, z0, t, atol = 1e-10, rtol = 1e-10)

In [18]: x = z[:,0]
         y = z[:,1]
         r = numpy.sqrt(x**2 + y**2)

         fig = pyplot.figure(figsize=(12, 4))
         ax1 = fig.add_subplot(131)
         ax1.plot(t, x, label = r"$x$")
         ax1.plot(t, y, label = r"$y$")
         pyplot.legend()
         ax1.set_xlabel(r"$t$")
```

```
    ax2 = fig.add_subplot(132)
    ax2.plot(x, y)
    ax2.set_xlabel(r"$x$")
    ax2.set_ylabel(r"$y$")
    ax3 = fig.add_subplot(133)
    ax3.plot(t, r)
    ax3.set_xlabel(r"$t$")
    ax3.set_ylabel(r"$r$")
```

Out[18]: <matplotlib.text.Text at 0x10fe560f0>



1.3 Nonlinear roots

```
In [19]: s = scipy.optimize.brentq(lambda x: numpy.cos(x) - x, 0.0, 1.0)
         print(s)
```

```
0.7390851332151559
```