Solving Equations & ODEs

FOSSEE

1 Solving linear equations

Consider following sets of equations:

$$
\begin{aligned}
3x + 2y - z &= 1 \\
2x - 2y + 4z &= -2 \\
-x + \frac{1}{2}y - z &= 0
\end{aligned}
$$

The matrix representation is:

$$A * x = B$$

Where A is coefficient matrix(in this case 3x3)
B is constant matrix(1x3)
x is the required solution.

```
In []: A = array([[3,2,-1], [2,-2,4], [-1, 0.5, -1]])
In []: B = array([[1], [-2], [0]])
In []: x = solve(A, B)
```

Solve the equation $Ax = B$ for x.
To check whether solution is correct try this:

```
In []: Ax = dot(A,x)  #Matrix multiplication of A and x(LHS)
In []: allclose(Ax, B)
Out[]: True
```

`allclose` Returns `True` if two arrays(in above case Ax and B) are element-wise equal within a tolerance.

2 Finding roots

2.1 Polynomials

$$x^2 + 6x + 13 = 0$$

to find roots, pylab provides `roots` function.

```
In []: coeffs = [1, 6, 13] #list of all coefficients
In []: roots(coeffs)
```

2.2 functions

Functions can be defined and used by following syntax:

```
def func_name(arg1, arg2):
    #function code
    return ret_value
```

A simple example can be

```
def expression(x):
    y = x*sin(x)
    return y
```

Above function when called with a argument, will return $x\sin(x)$ value for that argument.

```
In [95]: expression(pi/2)
Out[95]: 1.5707963267948966

In [96]: expression(pi/3)
Out[96]: 0.90689968211710881
```

2.3 Roots of non-linear equations

For Finding the roots of a non linear equation(defined as $f(x) = 0$), around a starting estimate we use `fsolve`:

```
In []: from scipy.optimize import fsolve
```

`fsolve` is not a part of `pylab`, instead is a function in the **optimize** module of **scipy**, and hence we **import** it.
For illustration, we want to find roots of equation:

$$f(x) = \sin(x) + \cos(x)^2$$

So just like we did above, we define a function:

```
In []: def f(x):
   ....:         return sin(x)+cos(x)**2
   ....:
```

Now to find roots of this non linear equation, around a initial estimate value, say 0, we use `fsolve` in following way:

```
In []: fsolve(f, 0) #arguments are function name and estimate
Out[]: -0.66623943249251527
```

3 ODE

We wish to solve an (a system of) Ordinary Differential Equation. For this purpose, we shall use `odeint`.

```
In []: from scipy.integrate import odeint
```

`odeint` is a function in the **integrate** module of **scipy**.

As an illustration, let us solve the ODE below:

$\frac{dy}{dt} = ky(L - y)$, L = 25000, k = 0.00003, y(0) = 250

We define a function (as below) that takes y and time as arguments and returns the right hand side of the ODE.

```
In []: def f(y, t):
  ....      k, L = 0.00003, 25000
  ....      return k*y*(L-y)
  ....
```

Next we define the time over which we wish to solve the ODE. We also note the initial conditions given to us.

```
In []: t = linspace(0, 12, 61)
In []: y0 = 250
```

To solve the ODE, we call the `odeint` function with three arguments - the function `f`, initial conditions and the time vector.

```
In []: y = odeint(f, y0, t)
```

Note: To solve a system of ODEs, we need to change the function to return the right hand side of all the equations and the system and the pass the required number of initial conditions to the `odeint` function.

4 Links and References

- Documentation for Numpy and Scipy is available at:
  http://docs.scipy.org/doc/
- For "recipes" or worked examples of commonly-done tasks in SciPy explore:
  http://www.scipy.org/Cookbook/