Introductory hump analysis with `scipy()` Mathematical Programming with Python

MATH 2604: Advanced Scientific Computing 4
Spring 2025
Monday/Wednesday/Friday, 1:00-1:50pm

https://people.sc.fsu.edu/~jburkardt/classes/python_2025/humps/humps.pdf

---

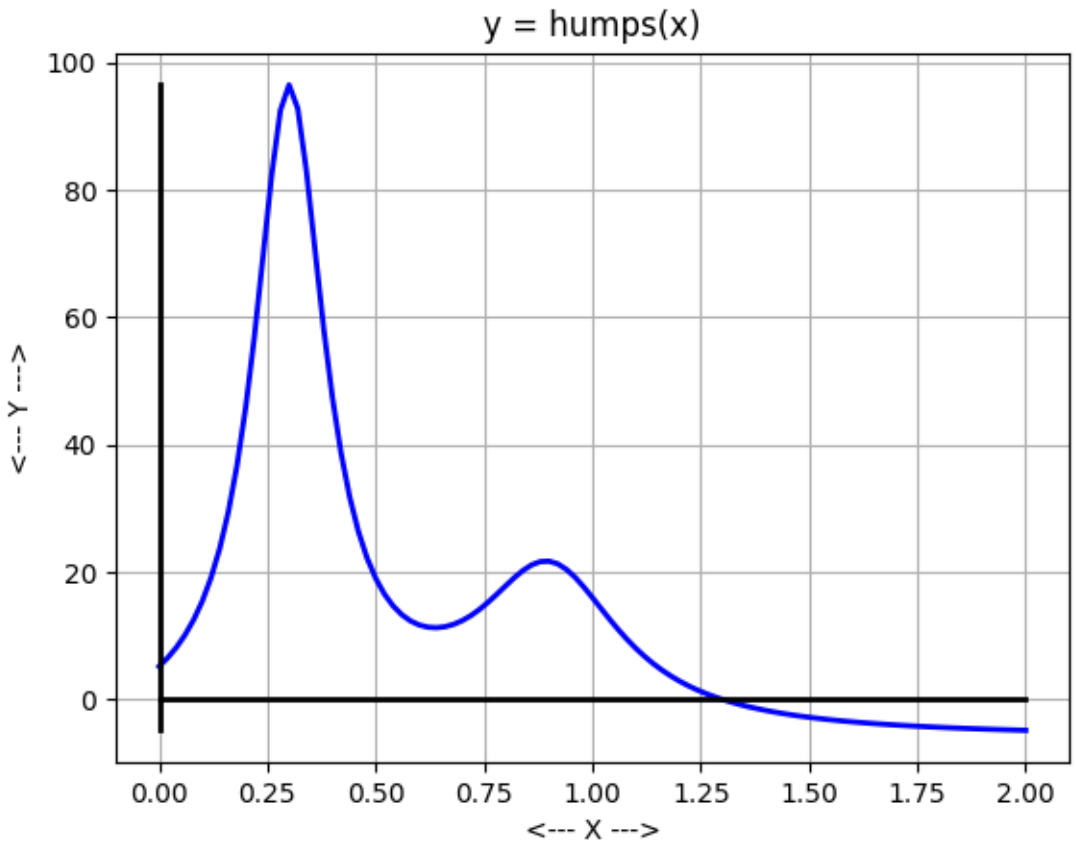
1 A Python library for scientific computation

The `scipy` library adds functions that are needed for computations across fields such as biology, chemistry, engineering, physics, statistics.

In most cases, `scipy` work with arrays from `numpy` and hence are vectorized. They often rely on underlying code compiled in Fortran, C, or C++, and hence are highly optimized.

Today we will demonstrate some `scipy` functions by working on a test examples made famous by MATLAB, known as `humps(x)`. This function is defined mathematically as:

$$y(x) = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6$$



The humps(x) function for $0 \leq x \leq 2$.

We will generally focus on this function over the interval $0 \leq x \leq 2$.

We will try to use `scipy` to investigate properties of this function. From a plot, we can see that the function seems to have a zero near $x = 1.25$, a local minimum near $x = 0.6370$, and two local maximum values near $x = 0.3$ and $x = 0.9$. The function values at the endpoints are $y(0) = 5.1764...$ and $y(2) = -4.8551...$. The integral of the function over $[0, 2]$ is approximately 29.3262... We will now look at how, instead of guessing from a plot, we could determine this information by calling the appropriate `scipy` functions.

The file `humps.py` will contain functions we will find useful during this work:

- `humps_antideriv(x)`: antiderivative function;
- `humps_deriv(x)`: first derivative;
- `humps_deriv2(x)`: second derivative;
- `humps_fun(x)`: evaluates humps(x);
- `humps_ode(x,y)`: like humps_deriv(), but includes y as second argument.

2 Where does `humps(x)` have a local minimum?

The `scipy.optimize` function `minimize_scalar()` is given a function $f(x)$ of a single variable, and seeks a value x for which $f(x)$ attains a locally minimum value. The search may be restricted to an interval, or

allowed to wander outside an initial pair of starting values.

We will seek the location and value of the local minimum of `humps(x)` within the interval $0.3 \leq x \leq 0.8$.

Here is a brief code that sets up the problem and requests a minimizer:

```
def humps_local_min ( ):

  from humps import humps_fun
  from scipy.optimize import minimize_scalar
  import numpy as np

  result = minimize_scalar ( humps_fun, bounds = [0.3, 0.8] )
  x = result.x
  y = humps_fun ( x )
  print ( '  Minimizer x = ', x )
  print ( '  Minimum value humps(x) = ', y )
  return
```

3 Where does humps(x) have a local maximum?

From the plot, we can see that `humps(x)` attains two local maximums in our interval of interest, near $x = 0.3$ and $x = 0.9$. Unfortunately, `scipy` does not include a corresponding "maximize_scalar()" function, so we have to improvise. The local maximum of a function $f(x)$ is a local minimum of the function $-f(x)$, so all we have to do is adjust our intervals, and define a Python function `humps_minus_fun()` which returns the negative of the original function. You have to be a little careful when you do this. Once you've got things correct, you should be able to locate both maximizers and the maximum values attained.

4 Given humps(x) data, interpolate a full curve

The `scipy.interpolate` library is given a list of pairs of data values $\{x_i, y_i\}$, and is asked to construct an interpolating function $f(x)$ such that $f(x_i) = y_i$. The interpolating function may be a spline, a piecewise function whose parts are of degree 0, 1, 2, or 3, or a polynomial. Interpolation can also be done in higher dimensions, where, for example, a function $f(x, y) = z$ is desired which matches data triples $\{x_i, y_i, z_i\}$.

The function `interp1d()` will be used to illustrate a few of the options for spline interpolation in the 1D case. We only supply 8 values of the `humps()` function as data. We then request interpolants of type 'nearest', 'linear', and 'cubic', and plot them as well as our known function.

```
def humps_interp ( ):

  from humps import humps_fun
  from scipy.interpolate import interp1d
  import matplotlib.pyplot as plt
  import numpy as np

  x = np.linspace ( 0.0, 2.0, 8 )
  y = humps_fun ( x )

  humps_nearest = interp1d ( x, y, kind = 'nearest' )
  humps_linear = interp1d ( x, y, kind = 'linear' )
  humps_cubic = interp1d ( x, y, kind = 'cubic' )

  x2 = np.linspace ( 0.0, 2.0, 101 )

  plt.plot ( x, y, 'o', label = 'data points' )
  plt.plot ( x2, humps_fun(x2), label = 'exact' )
  plt.plot ( x2, humps_nearest(x2), label = 'nearest' )
```

```
    plt.plot ( x2, humps_linear(x2), label = 'linear' )
    plt.plot ( x2, humps_cubic(x2), label = 'cubic' )
    plt.grid ( True )
    plt.legend ( )
    plt.show ( )
    return
```