Computational Physics Weekly Assessments

Week 6: gradient descent

Weekly problem task

This week you you will produce a plot and text illustrating the different behaviours of the gradient descent method when finding the minima of "Rosenbrock's Banana Function". You are expected to use your experience from the course and the previous 5 weekly assessments to help you structure your program, so less detailed instructions are given. The function to minimise is given by $f(x,y) = (1-x)^2 + 100(y-x^2)^2$.

Your script this week should produce a single plot and a variable "`ANSWER1`" that address…

Question 1: Describe how the behaviour of the GD method changes with different values of the step size parameter, gamma. Illustrate this by plotting example GD trajectories as required.

Question 2: Where does the minimum of the function occur? This question is to be answered by the result of your best GD trajectory. Do so by generating a variable called "`ANSWER2`", e.g. if the variables '`xmin`' and '`ymin`' are found by your code to represent the deepest minimum:

```
ANSWER2 = 'Minima occurs at %.2f, %.2f' % (xmin, ymin))
```

(To recap from the L1 Introduction to Programming course, in this context the % symbol is a format specifier. Each `%.2f` in the string is a *placeholder* for a floating point (non-whole) number with two decimal places. The variables in brackets after the isolated `%` are then substituted into the string when the code runs. You may wish to check your answer by putting a "`print ANSWER2`" statement in your code. The variable must be in the global scope of the module (i.e. NOT within a function.)

When run your script should produce a single plot that shows a 2D graphical representation of the function (as covered in week 6 lecture notes), with the GD trajectories drawn on the plot as well.

Hints

As a minimum, you will need to implement several functions as shown below:

- The function '`f((x, y))`' to be minimised.
- The function '`grad((x, y))`', which returns the vector gradient of the function. Use the analytical derivative (calculated "on paper").
- A function that takes an initial point and a gamma parameter and performs a number of iterations of the gradient descent method from that point, returning the trajectory.

You are free to use what bounds and GD starting point you wish. Bounds of -0.2<x<1.2 and -0.2<y<1.2 are suitable, as is a starting point of x=0.2 and y=1.0 You will likely need to call the GD function at least 3 times with different gamma values. You may wish to use a "for loop" for this – consider how you evaluated different panel counts in CP2 or different sample sizes in CP5. If you use the best value of gamma last, then you can simply access the last point of the last trajectory for `ANSWER2` after the for loop with, e.g. `xBest, yBest = trajectory[-1] # -1 means last row in array`

All the techniques that you need have been covered in previous weeks or the 6th lecture, so consider your assessments, the model solutions and the lecture notes. For example:

- Plotting of the GD trajectories is very similar to the plotting of the cannonball trajectories orbit in the 4th assessment.
- Conceptually – if not mathematically – the code is very similar to the DEQ and Euler solver for the pendulum, in that you update a vector position repeatedly using some function of that position to contribute to the update.
- You may find it helpful to use a 2-element numpy array as a mathematical vector to represent position, gradient etc. As with the Euler solver for the cannonball, this allows you to write code using vectors in some functions (in your GD method here, in the solve_euler for the cannonball) whilst accessing individual elements in other functions (f or grad here, your DEQ for the cannonball). Example code for both approaches are given below.
- Consider how many gradient descent steps to make – you may wish to iterate over a fixed number with a for loop, and you may further wish to consider using some convergence criteria as covered in the week 6 slides on DUO. How accurately do you need to measure the location of the minma?

from the lecture notes

Here are examples how to write functions for vector quantities stored in a numpy arrays:

File Edit Format Run Options Windows Help

```
def f(r):
    # Function of vector r to be minimised
    x, y = r

def grad(r):
    # Measure the vector gradient of the
    # function 'f' at position 'r'
    x, y = r

def gradient_descent(r):
    # Use the gradient descent technique to
    # minimise the function 'f' from a starting
    # point of 'r'
    x, y = r
```

Ln: 13 Col: 18

File Edit Format Run Options Windows Help

```
def f((x,y)):
    # Function of vector r to be minimised


def grad((x,y)):
    # Measure the vector gradient of the
    # function 'f' at position 'r'


def gradient_descent((x,y)):
    # Use the gradient descent technique to
    # minimise the function 'f' from a starting
    # point of 'r'
```

Ln: 10 Col: 23

IMPORTANT NOTE - The dynamic range of values of the function within the suggested bounds is very large, leading to an image that is lacking in contrast. The use of a logarithmic colour scale helps address this – see the example code below. You may need to change the black level (vmin) and white level (vmax) for different bounds.

rosen6.py – /Users/cds/cds/Teaching/2014-2015/L2_CompPhys/Week 6/rosen6.py

```
import matplotlib.colors # THIS BELONGS AT THE START OF YOUR SCRIPT
# Show a greyscale colourmap of the data
im = pyplot.imshow( dat, extent=(x0, x1,y0, y1), origin='lower',
                    cmap=matplotlib.cm.gray,
                    norm=matplotlib.colors.LogNorm(vmin=0.01, vmax=200))
pyplot.colorbar(im, orientation='vertical')
```

Ln: 39 Col: 0