AME

Week 7: Numerical Optimisation

Sophie Bindslev, November 2022

UNIVERSITY OF COPENHAGEN

Today's Plan

- Motivation
- Numerical Optimisers
- Newton Raphson: a gradient based optimiser
- Your time to shine!

Estimators as solutions to optimisation problems

- Most estimation problems involve or can be rewritten as maximising (or minimising) some objective function
- Common objective functions include the sum of squared residuals (OLS, NLS), the sum of absolute deviations (LAD) and log likelihoods
- For instance, the Non-Linear Least Squares estimator is the solution to the minimisation problem:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{argmin} \sum_{i=1}^{N} (y_i - m(\mathbf{x}_i\boldsymbol{\theta}))^2 \tag{1}$$

where $m(.)$ can be some nonlinear function of $\mathbf{x}_i\boldsymbol{\theta}$

- Sometimes analytical solutions to these optimisations problems are not readily available. Then numerical optimisers come in handy

Numerical Optimisers

- Generally speaking there are two types of numerical optimisers: gradient based and non gradient based
- **Gradient based**: faster but require your objective function to be smooth
- **Non gradient based**: in some sense more robust (can handle less smooth objective functions) but often slower
- Sometimes a combination of the two works well: start with a non-gradient based when far from the optimum and switch to a gradient based one once closer to the optimum (as measured by smaller step sizes, say)
- You can help your optimiser the more you do analytically beforehand, e.g. by ”feeding it” analytical Jacobians, Hessians

Gradient Based Optimisers

- The Newton Raphson (N-R) Algorithm is gradient based
- N-R approximates the objective function $f(x)$ with a 2nd order Taylor expansion. **Why is that a good idea?** The scalar case:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + 1/2f''(x_0)(x - x_0)^2 \qquad (2)$$

- N-R finds the minimum of this Taylor expansion at each iteration. Differentiating (2) wrt x and setting equal to zero:

$$0 = f'(x_0) + 2/2f''(x_0)(x - x_0) \Rightarrow \qquad (3)$$
$$x = x_0 - f'(x_0)/f''(x_0) \qquad (4)$$

- At each iteration i, N-R yields x_{i+1} by updating x_i using $f'(x_i)/f''(x_i)$. **What is the role of the slope of $f(x)$? What about the curvature?**

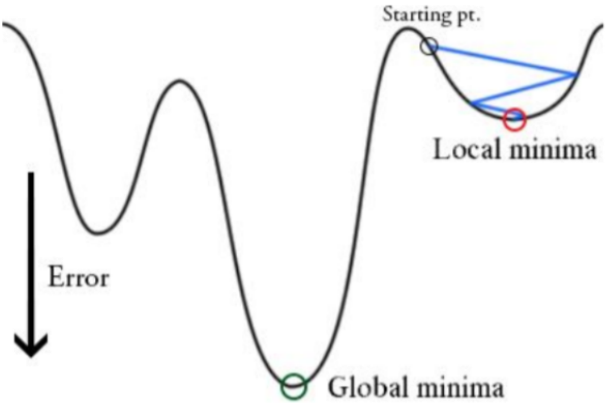
Matrix Notation

- Matrix version of the Newton-Raphson algorithm:

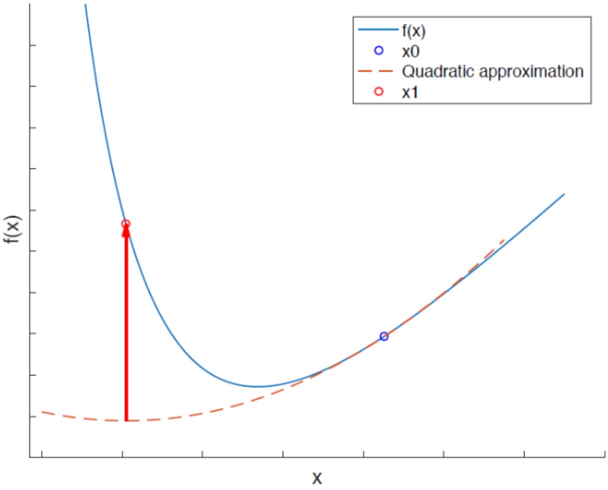
$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \mathbf{g}(\boldsymbol{\theta}_i)\mathbf{H}(\boldsymbol{\theta}_{\boldsymbol{i}})^{-1} \tag{5}$$

  where **H** and **g** are the Hessian and Gradient of the objective function, respectively.

Issues with NR I: Can get stuck in local extrema



Issues with NR II: Overshooting



Briefly on `lambda` functions

- You'll be using `lambda` to define functions. Previously, you've used `def ... return` when making your own functions
- Briefly speaking, `lambda` functions differ from normal functions in a number of ways, they
  - are a (slightly) faster way to write very short functions
  - are anonymous which means that if/when you get an error message it will refer to the function as `lambda`, not the function name
  - cannot contain statements e.g. assert, raise and so on
  - cannot contain type hints. When defining a function as
    $\texttt{def} \quad \texttt{f}(\texttt{x}:\texttt{np.ndarray}, \texttt{y}:\texttt{np.ndarray}) - - > \texttt{np.ndarray}$
    we've specified that our function takes numpy arrays as inputs and returns a numpy array. This can be helpful for reading code though you don't get an error message if you input the wrong type. `lambda` functions don't have this functionality

Your time to shine!

- Solve the problem set
- The `estimation.py` file gives you a function for computing the forward difference of a function i.e.

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} \tag{6}$$

  This can be used to approximate first and second derivatives if you don't want to (or can't) find these analytically
- You can watch the videos (linked in the problem set) if you'd like extra information