Numerical Methods in Scientific Computing, spring 2023

Problem sheet 8: minimization

Return by Monday 20.3.2023 23:59 to Moodle.

Bonus exercises are voluntary: they don't count towards the maximum for the exercises. They do, however, give you extra points in the grading of the course – it's possible to score over 100% of the points on the course with bonus exercises. No solutions will be provided for the bonus exercises.

1. (pen and paper, 2 points) Draw a mind map on the material discussed this week (minimization). What kind of grouping can you think for the topics? How do the topics link together? Make sure to go read through the lecture notes again in detail so that you include all methods and algorithms.

2. (pen and paper, 2 points) Newton's method for function minimization uses information on the local curvature of the function $f(s)$ to obtain an improved estimate for the extremum $s \to s + \Delta s$ as

$$\Delta s = -\frac{f'(s)}{f''(s)}.$$

   (Note that this is the same as Newton's method for root finding, applied to $f'(s) = 0$.) Contrast this method with the parabola fitting method, which is based on information on $f(a)$, $f(b)$, $f(c)$:

$$\Delta s = -\frac{1}{2}\frac{(s-a)^2[f(s)-f(c)] - (s-c)^2[f(s)-f(a)]}{(s-a)[f(s)-f(c)] - (s-c)[f(s)-f(a)]}$$

   where we have set $b = s$.

   Both methods approximate the function $f(s)$ with a parabola $f(s) = as^2 + bs + c$. What is the difference between them? Comment on this difference.

   Can you think of a case where one method works but the other one fails?

3. (pen and paper, 4 points) Your friend is studying a line minimization problem $f(\lambda)$, in which evaluating $f(\lambda)$ is very expensive. However, the derivative $f'(\lambda)$ is cheap to calculate with $f(\lambda)$, and a theoretical upper limit for a maximal sensible step size $\lambda_{\max}$ has been derived for the problem in the literature.[1]

   Fit a parabola $f(\lambda) = a\lambda^2 + b\lambda + c$ given values for $f(0)$, $f'(0)$, $f(\lambda_{\max})$. Find the minimizer λ^* of $f(\lambda)$, expressing it in terms of the given input values $f(0)$, $f'(0)$, $f(\lambda_{\max})$.

4. (computer, 6 points) Implement the golden section search algorithm (don't use any library implementations thereof!) for the function

$$f(x) = x^4 \text{erfc}\, x - \sin x$$

   in the search bracket $x^* \in [x_a, x_c]$, where erf is the error function and $x \geq 0$. Organize your code in a function `golden(xa,xc)` that returns the position of the minimum.

   Check that your code works by finding the minimum of the function. Start out by plotting the function to define a reasonable search bracket $[x_a, x_c]$.

[1] For those interested, the context here is the optimization of one-particle states in electronic structure theory: the energy depends on the single-particle states $|\psi_i\rangle$, which are optimized with unitary transformations $|\psi_i\rangle \to \sum_j |\psi_j\rangle(\exp \boldsymbol{\theta})_{ji}$ where $\boldsymbol{\theta}^\dagger = -\boldsymbol{\theta}$ is an antihermitian matrix of rotation parameters. Evaluating $E(\boldsymbol{\theta})$ is expensive, but the evaluation often also yields the information to compute the gradient $\partial E/\partial\theta_{pq}$. These optimization problems are often very large: the simple case of a Hartree–Fock or density functional calculation only requires rotations between occupied (o) and unoccupied (v) orbitals. A moderate size calculation may have $o = 100$ occupied orbitals and $v = 1000$ unoccupied orbitals, yielding 10^5 optimization parameters.

5a. (computer, 6 points) Write a function `minimize(x0, y0, alg)` that minimizes

$$f(x,y) = 7e^{-(x-2)^2} + 3x^2 + 2xy + 5y^2 - 8e^{-(y+4)^2}$$

using the Nelder–Mead simplex method (`alg=1`), steepest descent (`alg=2`), Polak–Ribière conjugate gradients (`alg=3`), or the Broyden–Fletcher–Goldfarb–Shanno algorithm (`alg=4`).

If you use C, you can use the routines in the GNU Scientific Library (GSL) to implement the minimization.

If you use Python, you can use the routines in SciPy. Because SciPy does not have steepest descent, use modified Powell's method instead (`fmin_powell`).

5b. (computer, 4 points) Study the behavior of the algorithms with the starting points $(x, y) = (20, -10)$, and $(x, y) = (2, -4)$. How many minima do you find? Which algorithm converges the fastest? Comment on the results.

6. (BONUS, pen and paper, 6 points) Can you extend the treatment of problem 3 to a third-order polynomial $y(\lambda) = a\lambda^3 + b\lambda^2 + c\lambda + d$ fit to $y(0)$, $y'(0)$, $y(\lambda_{\max})$, and $y'(\lambda_{\max})$? Give explicit expressions for a, b, c, and d. The function $y'(\lambda)$ is now a parabola with up to two roots on the real axis. Which one should you pick as λ^*? Justify your choice with a mathematical argument.

Note: this approach can be used for an analytic line search $y(\lambda) = f(\boldsymbol{x} + \lambda\boldsymbol{s})$ where $\boldsymbol{s}$ is the search direction. The values used for the fit can be evaluated as

$$\begin{aligned} y(0) &= f(\boldsymbol{x}) \\ y'(0) &= \boldsymbol{s}^{\mathrm{T}} \nabla f(\boldsymbol{x}) \\ y(\lambda_{\max}) &= f(\boldsymbol{x} + \lambda_{\max}\boldsymbol{s}) \\ y'(\lambda_{\max}) &= \boldsymbol{s}^{\mathrm{T}} \nabla f(\boldsymbol{x} + \lambda_{\max}\boldsymbol{s}) \end{aligned}$$

where $\nabla f(\boldsymbol{x})$ means evaluating ∇f at point $\boldsymbol{x}$.