H5: Optimization

5.1 introduction

5.1.1 concept and notation

concept of optimization

= problem of determining an argument for which a given function has an extreme value on a given domain

formally: given a function $f : \mathbb{R}^n \to \mathbb{R}$, and a set $S \subseteq \mathbb{R}^n$, we seek $\mathbf{x}^* \in S$ such that f attains a minimum on S at $\mathbf{x}^*$, i.e. $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in S$.

objective function f

= the function f for which we want to find an extremum

> may be linear or non-linear, but is always differentiable

feasible set S

= set of equations and inequalities which put constraints on f

> any x which satisfies the constraints is called a feasible point, ie x∈S

If S = $\mathbb{R}^n$, the problem is unconstrained

types of optimization problems

A continuous opt. prob. has the form:

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{subject to} \quad \mathbf{g}(\mathbf{x}) = \mathbf{0} \quad \text{and} \quad \mathbf{h}(\mathbf{x}) \leq \mathbf{0}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$ and $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^p$.

is f, g and h are linear, the opt. problem is called linear programming
if not, its nonlinear programming

5.2 optimality conditions

5.2.1 unconstrained optimality conditions

first-order necessary condition

We have a minimum in x* if:

$$\nabla f(\mathbf{x}^*) = 0$$

> is a necessary condition, but doesn't prove x* is a minimum
ie: it can be a maximum, minimum or saddle point

second-order sufficient condition

Consider the Hessian:

$$\mathbf{H}_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{bmatrix}$$

Now: At a critical point $\mathbf{x}^*$, where $\nabla f(\mathbf{x}) = \mathbf{0}$, if $\mathbf{H}_f(\mathbf{x}^*)$ is...

- Positive definite, then $\mathbf{x}^*$ is a minimum of f
- Negative definite, then $\mathbf{x}^*$ is a maximum of f
- Indefinite, then $\mathbf{x}^*$ is a saddle point of f
- Singular, then various pathological situations can occur

5.3 optimization in one dimension

unimodal function

= a function f:$\mathbb{R}\to\mathbb{R}$ on interval [a,b] for which there is a unique x*
AND for any x_1,x_2: $x_2<x^*$ implies $f(x_1)>f(x_2)$ and $x_1>x^*$ implies $f(x_1) < f(x_2)$

ie: f(x) is strictly increasing for x<x* and strictly increasing for x>x*

5.3.1 golden section search	
golden section search	For **f** unimodal on [a,b] For $x_1,x_2 \in [a,b]$ with $x_1<x_2$ > compare the function values $f(x_1)$ and $f(x_2)$ > exclude either $[x_2,b]$ or $[a,x_1]$: - $f(x_1)< f(x_2) \rightarrow$ minimum on $[a,x_1]$ - $f(x_1)> f(x_2) \rightarrow$ minimum on $[x_2,b]$ > repeat this until you find the minimum To optimize this, each pair of points has the same relative position as the old pair > choose the this distance as τ = golden ratio = $(\sqrt{(5)}-1)/2 \approx 0.618$ and 1-τ >> will converge if function is unimodal within initial bracket
5.3.2 successive parabolic search	
successive parabolic search	evaluate the function at three points > fit a parabola to the points > use the minimum of the parabola as a new approximate value of the minimum >> algorithm isn't guaranteed to converge
5.3.3 Newton's method	
Newton's method	Because of the Taylor expansion, the function is locally approximated by: $$f(x+h) \approx f(x) + f'(x)h + \frac{1}{2}f''(x)h^2$$ thus the minimum is given by: $$-f'(x)/f''(x),$$ use this ti find the minimum of the objective function in an iterative way >> only converges if it started sufficiently close to the minimum > might converge to a maximum or inflection point
5.4 multidimensional unconstrained optimization	
5.4.1 direct search	
direct search	analogous to golden section search ie: objective function values are compared to one another > however isn't guaranteed to converge
ex: Nelder and Mead	for a function f: $\mathbb{R}^n \rightarrow \mathbb{R}$, choose n+1 points that form a simplex ie: n+1 points that aren't colinear > generate a new point alon the straight line connecting the point with the highest function value and the centroid of the remaining n points > this new point replaces the other one >> repeat until converge
5.4.2 steepest descent	
steepest descent	if we go in a direction -∇f(**x**) we go into a descending direction > thus for a fixed point **x** and direction **s** = -∇f(**x**), define $$\phi(\alpha) = f(\mathbf{x}+\alpha\mathbf{s})$$ > now we have a one-dimensional optimization problem > can be solved using previous methods >> repeat until converge

5.4.3 Newton's method	
Newton's method	Approximate via Taylor: $$f(\mathbf{x}+\mathbf{s}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T\mathbf{s} + \frac{1}{2}\mathbf{s}^T\mathbf{H}_f(\mathbf{x})\mathbf{s}, \quad (1$$ where $\mathbf{H}_f(\mathbf{x})$ is the *Hessian matrix*. This quadratic function in $\mathbf{s}$ is minimized when $$\mathbf{H}_f(\mathbf{x})\mathbf{s} = -\nabla f(\mathbf{x}). \quad (1$$ > unreliable method, unless started close enough to the solution
5.4.4 quasi-Newton methods	
quasi-Newton methods	to lower calculations per iteration, consider using a general form: $$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{B}_k^{-1}\nabla f(\mathbf{x}_k)$$ with α_k a line search parameter $\mathbf{B}_k$ some approximation of the Hessian matrix
5.4.5 secant updating methods	
secant updating methods	methods which preserve symmetry in the approximate Hessian and positive definiteness > reduces work and storage by half + quasi-Newton step will be in descent >> most effective: BFGS (see course)
5.4.6 conjugate gradient method	
conjugate gradient method	= alternative to Newton's method that doesn't use second derivatives > doesn't even use an approximation to Hessian matrix > suitable for large matrices uses gradient, but avoids repeatedly searching in the same directions > modifies the new gradient at each step to remove components in previous directions > sequence of conjugate: $(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{H}_f\mathbf{y}$
5.5 nonlinear least squares	
nonlinear least squares problem	view the problem as a special case of nonlinear optimization > given data points (t_i,y_i), i=,…,m we wish to find a vector $\mathbf{x}\in\mathbb{R}^n$ that best fits the model: $$f(t, \mathbf{x}), \text{ where } f : \mathbb{R}^{n+1} \to \mathbb{R}.$$ > define the components of the residual function $\mathbf{r}:\mathbb{R}^n\to\mathbb{R}^m$ by: $$r_i(\mathbf{x}) = y_i - f(t_i, \mathbf{x}), \quad i = 1, \ldots, m,$$ then we wish to minimize the function $$\phi(\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x})^T\mathbf{r}(\mathbf{x})$$ Apply Newton's method: $\mathbf{x}_k$ is an opproximate solution > $\mathbf{s}_k$ is given by the linear system: $$\mathbf{H}_\phi(\mathbf{x}_k)\mathbf{s}_k = -\nabla\phi(\mathbf{x}_k)$$ where the gradient vector and Hessian matrix of ϕ are given by $$\nabla\phi(\mathbf{x}) = \mathbf{J}^T(\mathbf{x})\mathbf{r}(\mathbf{x})$$ and $$\mathbf{H}_\phi(\mathbf{x}) = \mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) + \sum_{i=1}^{m} r_i(\mathbf{x})\mathbf{H}_{r_i}(\mathbf{x})$$ in which $\mathbf{J}^T(\mathbf{x})$ is the Jacobian matrix of $\mathbf{r}(\mathbf{x})$, and $\mathbf{H}_{r_i}(\mathbf{x})$ denotes the Hessian matrix of the component function $r_i(\mathbf{x})$. The Newton step $\mathbf{s}_k$ is thus given by the linear system $$\left(\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x_k}) + \sum_{i=1}^{m} r_i(\mathbf{x}_k)\mathbf{H}_{r_i}(\mathbf{x}_k)\right)\mathbf{s}_k = -\mathbf{J}^T(\mathbf{x}_k)\mathbf{r}(\mathbf{x}_k)$$

5.5.1 Gauss-Newton method	
Gauss-Newton method	note that $\mathbf{H}_{ri}$ is multiplied by the corresponding residual component r_i > should be really small at a solution > drop this term: $$\left(\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x_k})\right)\mathbf{s}_k = -\mathbf{J}^T(\mathbf{x}_k)\mathbf{r}(\mathbf{x}_k)$$ determines an approximate Newton step $\mathbf{s}_k$ at each iteration. Recognize this system as the normal equations for the mxn linear least squares problem: $$\mathbf{J}(\mathbf{x}_k)\mathbf{s}_k \cong -\mathbf{r}(\mathbf{x}_k)$$ which can be solved more reliably by orthogonal factorization of $\mathbf{J(x_k)}$ > next approximate solution is given by: $\mathbf{x_{k+1}} = \mathbf{x_k} + \mathbf{s_k}$ > repeat until convergence
5.5.2 Levenberg-Marquardt method	
Levenberg-Marquardt method	at each iteration, the linear system for the step s_k is of the form: $$\left(\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x_k}) + \mu_k\mathbf{I}\right)\mathbf{s}_k = -\mathbf{J}^T(\mathbf{x}_k)\mathbf{r}(\mathbf{x}_k)$$ with μ_k a nonnegative scalar parameter chosen by some stategy > corresponding linear least squares problem is: $$\begin{bmatrix}\mathbf{J}(\mathbf{x}_k)\\ \sqrt{\mu_k}\,\mathbf{I}\end{bmatrix}\mathbf{s}_k \cong \begin{bmatrix}-\mathbf{r}(\mathbf{x}_k)\\ \mathbf{0}\end{bmatrix}$$
5.6 constrained optimization	
constrained optimization problem	= optimization problem for which the minimum is located outside the feasible set > solution occurs on the boundary of the feasible set They can be linear and nonlinear and subdivided into two categories: • **Equality constraints** which are of the general form $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ • **Inequality constraints** which are of the general form $\mathbf{h}(\mathbf{x}) \leq \mathbf{0}$ >> can be solved using Lagrange multipliers
5.6.1 the trust-region constrained algorithm	
the trust-region constrained algorithm	The minimize function of *scipy.optimize* provides several algorithms for constrained min > one of those is the t-r constr. alg. which deals with constr. min. problems of the form: $$\min_{\mathbf{x}} f(\mathbf{x})$$ $$\text{subject to:} \quad \begin{aligned}\mathbf{c}_l \leq \mathbf{c}(\mathbf{x}) \leq \mathbf{c}_u\\ \mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u\end{aligned}$$ When $c_{l,j} = c_{u,j}$ the method reads the j^{th} constraint as an equality constraint and deals with it accordingly. Beside that, one-sided constraints can be specified by setting the upper (u) or lower (l) bound to `np.inf` with the appropriate sign. As an illustration this method will be applied to the Rosenbrock example.