Technical University Berlin
Faculty II – Mathematics and Natural Sciences
Modeling, Numerics, Differential Equations Group
Research Group Numerical Mathematics of Nonlinear Optimization
Prof. Dr. Dietmar Hömberg

Automatic Differentiation

Seminar Summary

April, 2019

Daniel Runge
daniel.benjamin.runge@gmail.com

Contents

1 Introduction

Automatic differentiation is an important tool for many disciplines in applied mathematics. Many basic problems such as the approximation of solutions to nonlinear equations, derivative-based nonlinear optimization algorithms, the approximation of solutions to linear and nonlinear ordinary and partial differential equations and problems in the calculus of variations to name only a few can be handled quite well by calculating derivatives of certain functions to put them to use in a wide variety of numerical algorithms. Thus, fast and reliable methods for the automatic computation of derivatives represent an important building block in the discipline of scientific computing.

Concrete applications in engineering and science range from such diverse topics as minimizing the amount of building material to construct the roofs of buildings (minimal surface problem) to simulating the behaviour of thin-film electrochemical fuel cells (two-point boundary value problem) and even classification and fitting problems in machine learning using feedforward neural networks (backpropagation). The possible applications are truly boundless.

Automatic differentiation can broadly be divided into two classes: *finite difference approximation* and exact *algorithmic differentiation*. Accordingly, this summary is also divided into two sections.

The first class of methods concerns itself with the derivation of approximation formulae of first and higher order derivatives based on Taylor's theorem. Their accuracy hinges on certain smoothness assumptions imposed upon the target function. This class finds a broad range of applications in optimization techniques and the numerical treatment of ordinary and partial differential equations which is the reason why section 2.2 specifically investigates the influence of approximating Jacobian matrices with finite differences on the convergence properties of Newton's method and section 2.4 gives a quick overview of the finite difference method to approximate solutions to two-point boundary value problems. An acceleration technique based on the graph colouring problem seeks to minimize the number of evaluations of the target function in approximating its Jacobian and is presented in section 2.1.2.

Colouring techniques can also be combined with algorithmic differentiation methods. Methods of this kind are able to compute the derivatives of the target function exactly by reducing any function to a composition of differentiable basis functions. There is a distinction between the forward (section 3.2) and reverse (section 3.3) modes which are in terms of computational complexity better suited for vector-valued or scalar functions respectively. We conclude the second part with a simplified exemplary minimization problem based on the minimal surface problem which is attacked with a simple line search gradient descent method. The proposed method was implemented in `python` and applied algorithmic differentiation tools provided by the `CasADi` [2] optimization library. A couple of approximated solutions are presented there and some basic examples are provided on my `GitHub` account[1].

I sincerely hope that you, dear reader, can find some use and joy in reading this little summary. If you happen to come across any mistakes, please do not hesitate to contact me. Happy reading!

[1] https://github.com/Anemometer/minimal_surface

2 Finite Difference Derivative Approximations

For finite difference approximations, the most useful tool in our toolbox – at least for higher order approximations – is going to be Taylor's theorem. Denoting by $D^j f(x_0)$ the $j-th$ derivative of f in some point x_0 and associating it in the usual way with a j-multilinear form [4, p.96], we obtain Taylor's theorem for the special case of finite-dimensional Banach spaces.

Theorem 2.1 (Taylor's Theorem, [4, p.508]). *Let V, W be finite-dimensional Banach spaces, $G \subseteq V$ open, $f : G \to W$ at least $k \in \mathbb{N}$ times differentiable, $x \in G$, and $p \in G$ such that $x + p \in G$.*

1. *For all $x \in G$ holds*

$$f(x+p) = \left(\sum_{j=0}^{k} \frac{1}{j!} D^j f(x) \underbrace{(p, \dots, p)}_{j \text{ times}} \right) + R(x+p), \tag{2.1}$$

   where $\lim_{p \to 0} \frac{R(x+p)}{\|p\|^k} = 0$.

2. *If f is $(k+1)$ times differentiable and scalar-valued, i.e. $W = \mathbb{R}$ and if $\overline{(x+p)x} := \{x + \lambda p \mid \lambda \in [0,1]\} \subseteq G$, then there exists some $\xi \in]0,1[$ such that*

$$R(x+p) = \frac{1}{(k+1)!} D^{k+1} f(x + \xi p) \underbrace{(p, \dots, p)}_{k+1 \text{ times}} \tag{2.2}$$

Assuming e.g. that G is convex and f is twice continuously differentiable, $f \in \mathcal{C}^2(G)$, Theorem 2.1 gives us the first-order estimate

$$\|f(x+p) - f(x) - Df(x)p\| = \frac{1}{2} \|D^2 f(x + \xi p)(p,p)\| \leq \frac{1}{2} \|D^2 f(x + \xi p)\| \|p\|^2, \tag{2.3}$$

where

$$\|D^2 f(x + \xi p)(p,p)\| := \sup_{\|v\|=1, \|w\|=1} \|D^2 f(x + \xi p)(v,w)\|$$

is the operator norm on the space of continuous bilinear mappings. Estimates on the accuracy of finite difference approximations of the gradient or the Jacobian of f can be made by assuming for instance that the operator norm of the second derivative is bounded by some constant $L > 0$ such that

$$\|f(x+p) - f(x) - Df(x)p\| \leq \frac{L}{2} \|p\|^2.$$

However, this requires f to be twice continuously differentiable. This requirement can be weakened somewhat to obtain the result without imposing smoothness beyond the first derivative.

Lemma 2.2. *Let $f : G \to W$ be differentiable, $G \subseteq \mathbb{R}^n$ open, $W \subseteq \mathbb{R}^m$, and Df Lipschitz-continuous with a Lipschitz constant of $L > 0$. Then for $x \in G$ and $p \in G$ such that $x + p \in G$, it holds*

$$\|f(x+p) - f(x) - Df(x)p\| \leq \frac{L}{2} \|p\|^2. \tag{2.4}$$

Proof.

Let $x, y \in G$ and write $f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix}$. Consider the real functions $g_k : [0,1] \to \mathbb{R}$, $\theta \mapsto f_k(x + \theta(y-x))$, $k \in \{1, \dots, m\}$ which are differentiable. By the chain rule, the derivative of g_k is given by

$$\begin{aligned} \frac{d}{d\theta} g_k(\theta) &= \frac{d}{d\theta} f_k(x + \theta(y-x)) \\ &= Df_k(x + \theta(y-x))(y-x) \\ &= [Df(x + \theta(y-x))_{i,j}]_{i=k}^{j=1,\dots,n}(y-x) \\ &= (Df(x + \theta(y-x))(y-x))_k \end{aligned} \tag{2.5}$$

where $(Df(x + \theta(y-x))(y-x))_k$ is the k-th entry of the vector $Df(x + \theta(y-x))(y-x)$. By the fundamental theorem of calculus, we obtain

$$\begin{aligned} f_k(y) &= f_k(x) + f_k(y) - f_k(x) \\ &= f_k(x) + \int_0^1 \frac{d}{d\theta} f_k(x + \theta(y-x))\, d\theta \\ &= f_k(x) + \int_0^1 (Df(x + \theta(y-x))(y-x))_k\, d\theta \qquad \text{(by (2.5))} \end{aligned} \tag{2.6}$$

and hence

$$\begin{aligned} f(y) &= \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix} + \begin{bmatrix} \int_0^1 (Df(x + \theta(y-x))(y-x))_1\, d\theta \\ \vdots \\ \int_0^1 (Df(x + \theta(y-x))(y-x))_m\, d\theta \end{bmatrix} \qquad \text{(by (2.6))} \\ &= f(x) + \int_0^1 Df(x + \theta(y-x))(y-x)\, d\theta \end{aligned}$$

where the integral of a vector-valued function is defined component-wise. Consequently,

$$\begin{aligned} f(y) - f(x) &= \int_0^1 Df(x + \theta(y-x))\, d\theta \\ &= \int_0^1 Df(x + \theta(y-x)) + Df(x)(y-x) - Df(x)(y-x)\, d\theta \\ &= Df(x)(y-x) + \int_0^1 [Df(x + \theta(y-x)) - Df(x)]\,(y-x)\, d\theta \end{aligned}$$

which implies

$$\|f(y) - f(x) - Df(x)(y-x)\| = \left\| \int_0^1 [Df(x + \theta(y-x)) - Df(x)]\,(y-x)\, d\theta \right\|. \tag{2.7}$$

Now, since the Riemann integral of every component function of

$$\theta \mapsto [Df(x + \theta(y-x)) - Df(x)]\,(y-x) =: h(\theta),$$

is continuous because Df is Lipschitz continuous, it is also integrable and can thus be written as the limit of a Riemann sum with respect to an equally spaced partition of $[0,1]$ [8, p.198]:

$$\theta_k = \frac{k}{n},\ k = 0, \dots, n, \quad n \in \mathbb{N}$$

$$\int_0^1 h(\theta)\,\mathrm{d}\theta = \lim_{n\to\infty} \sum_{k=1}^{n} \begin{bmatrix} h(\theta_k)_1 \\ \vdots \\ h(\theta_k)_m \end{bmatrix} (\theta_k - \theta_{k-1})$$

But the triangle inequality implies for *every* Riemann sum that

$$\left\| \sum_{k=1}^{n} \begin{bmatrix} h(\theta_k)_1 \\ \vdots \\ h(\theta_k)_m \end{bmatrix} (\theta_k - \theta_{k-1}) \right\| \leq \sum_{k=1}^{n} \left\| \begin{bmatrix} h(\theta_k)_1 \\ \vdots \\ h(\theta_k)_m \end{bmatrix} (\theta_k - \theta_{k-1}) \right\|$$

Due to the continuity of $\|\cdot\|$ and the resulting integrability of $\theta \mapsto \|h(\theta)\|$, we therefore see that

$$\left\| \int_0^1 h(\theta)\,\mathrm{d}\theta \right\| \leq \int_0^1 \|h(\theta)\|\,\mathrm{d}\theta. \tag{2.8}$$

Combining (2.8) and (2.7) produces

$$\begin{aligned} \|f(y) - f(x) - Df(x)(y-x)\| &= \left\| \int_0^1 [Df(x+\theta(y-x)) - Df(x)]\,(y-x)\,\mathrm{d}\theta \right\| \\ &\leq \int_0^1 \|Df(x+\theta(y-x) - Df(x))\|\,\|y-x\|\,\mathrm{d}\theta \qquad \text{(by (2.8))} \\ &\leq \int_0^1 L\|y-x\|^2\,\mathrm{d}\theta \qquad \text{(Df Lipschitz continuous)} \\ &= L\|y-x\|^2. \end{aligned}$$

Setting $y = x + p$ yields the claim (2.4). □

The estimate in (2.4) will be of great use in error estimates for Jacobian and gradient approximations as well as in the proof of the convergence of the Newton method both with exact and approximated Jacobians.

2.1 Approximation of First Order Derivatives

Given a differentiable function $f : G \to \mathbb{R}^m$ for $G \subseteq \mathbb{R}^n$ open and convex, our aim is to approximate the partial derivatives of f at some point $x \in G$ in order to approximate gradients, Jacobians or directional derivatives.

If we impose sufficiently strong smoothness assumptions onto f, different kinds of difference formulas can be extracted from Theorem 2.1. For $f \in \mathcal{C}^2(G)$ and for any $(x+p) \in G$ there exists some $t \in]0,1[$ such that

$$f(x+p) = f(x) + Df(x)p + \frac{1}{2}D^2 f(x+tp)(p,p).$$

Setting $p = \varepsilon e_i$ for some $\varepsilon > 0$ and e_i the i-th column of the identitiy matrix this leaves us with

$$f(x+\varepsilon e_i) = f(x) + \varepsilon \underbrace{Df(x)e_i}_{=\frac{\partial f}{\partial x_i}(x)} + \frac{1}{2}\varepsilon^2 D^2 f(x+t\varepsilon e_i)(e_i,e_i)$$

which, after some rearrangement, results in

$$\frac{f(x+\varepsilon e_i) - f(x)}{\varepsilon} = \frac{\partial f}{\partial x_i}(x) + \frac{1}{2}\varepsilon D^2 f(x+t\varepsilon e_i)(e_i,e_i).$$

If $\|e_i\| = 1$ and assuming an upper bound L on the operator norm of $D^2 f(x)$ in G, the remarks following (2.3) establish that

$$\left\| \frac{f(x + \varepsilon e_i) - f(x)}{\varepsilon} - \frac{\partial f}{\partial x_i}(x) \right\| \leq \frac{1}{2} L \varepsilon \tag{2.9}$$

which means that the *forward difference formula*

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + \varepsilon e_i) - f(x)}{\varepsilon} \tag{2.10}$$

is an approximation of the partial derivative $\frac{\partial f}{\partial x_i}(x)$ which converges linearly to the exact value as $\varepsilon \to 0$. Note however that Lemma 2.2, by virtue of (2.4), supplies us with the same result with the weaker assumption that $f \in \mathcal{C}^1(G)$ and Df being Lipschitz-continuous on G.

An even higher degree of precision can be attained by performing the Taylor expansion for both $p = +\varepsilon e_i$ and $p = -\varepsilon e_i$ and assuming $f \in \mathcal{C}^3(G)$:

$$\begin{aligned} f(x + \varepsilon e_i) &= f(x) + \varepsilon \frac{\partial f}{\partial x_i}(x) + \frac{1}{2}\varepsilon^2 D^2 f(x)(e_i, e_i) + o(\varepsilon^3) \\ f(x - \varepsilon e_i) &= f(x) - \varepsilon \frac{\partial f}{\partial x_i}(x) + \frac{1}{2}\varepsilon^2 D^2 f(x)(e_i, e_i) + o(\varepsilon^3) \end{aligned}$$

where $g(x) = o(h(x)),\ x \to a \iff \lim_{x \to a} \left\| \frac{g(x)}{h(x)} \right\| = 0$. Subtracting the second equation from the first gives us

$$2\varepsilon \frac{\partial f}{\partial x_i}(x) = f(x + \varepsilon e_i) - f(x - \varepsilon e_i) + o(\varepsilon^3).$$

Seeing as $o(\varepsilon^3)/\varepsilon = o(\varepsilon^2)$, rearranging this equation results in the *central difference formula*

$$\frac{\partial f}{\partial x_i}(x) = \frac{f(x + \varepsilon e_i) - f(x - \varepsilon e_i)}{2\varepsilon} + o(\varepsilon^2) \tag{2.11}$$

which converges quadratically to the precise value of $\frac{\partial f}{\partial x_i}(x)$ as $\varepsilon \to 0$. Besides these estimates for first order partial derivatives, we can also establish a similar result for the operator norm of the Jacobian so long as it is induced by a norm which satisfies $\|e_i\| = 1$ for $i \in \{1, \ldots, n\}$. For the sake of simplicity, we shall establish the bound for the operator norm induced by the 1-norm

$$\|x\|_1 := \max_{i \in \{1, \ldots, n\}} |x_i|.$$

Since all norms in finite dimensional vector spaces are equiavalent the same results holds true (with modified constants) for any operator norm so long as it is induced by a norm satisfying $\|e_i\| = 1$. Given a vector $h \in \mathbb{R}^n$, we denote the forward difference approximation of the Jacobian $Df(x)$ in a point $x \in G$ by $\Delta_h f(x)$ where

$$\Delta_h f(x) = [\Delta_{h,j} f(x)]_{j=1}^n = \left[\frac{f(x + h_j e_j) - f(x)}{h_j} \right]_{j=1}^n.$$

Theorem 2.3 (Lemma 3.2.1 in [10]). *Let* $f : G \to \mathbb{R}^m$, $G \subseteq \mathbb{R}^n$ *open with* Df *Lipschitz-continuous with a constant* $L > 0$, $h \in \mathbb{R}^n$ *such that* $x, x + h_j e_j \in G$ *for* $j \in \{1, \dots, n\}$ *and* $\|\cdot\|$ *be a norm such that* $\|e_i\| = 1$ *for* $i \in \{1, \dots, n\}$. *Then it holds*

$$\|\Delta_{h,j} f(x) - Df(x)e_j\| \leq \frac{L}{2}|h_j| \tag{2.12}$$

Additionally, for $\|\cdot\| = \|\cdot\|_1$, *we have for the induced operator norm*

$$\|\Delta_h f(x) - Df(x)\|_1 \leq \frac{L}{2}\|h\|_\infty \tag{2.13}$$

Proof.

First, observe that

$$\begin{aligned}\|\Delta_{h,j} f(x) - Df(x)e_j\| &= \left\| \frac{f(x + h_j e_j) - f(x)}{h_j} - Df(x)e_j \right\| \\ &= |h_j|^{-1} \left\| f(x + h_j e_j) - f(x) - Df(x)h_j e_j \right\|.\end{aligned}$$

Lemma 2.2 then implies $\left\| f(x + h_j e_j) - f(x) - Df(x)h_j e_j \right\| \leq \frac{L}{2}\|h_j e_j\|^2$. But since $\|e_j\| = 1$, we obtain

$$\|\Delta_{h,j} f(x) - Df(x)e_j\| \leq |h_j|^{-1} \frac{L}{2}\|h_j e_j\|^2 = \frac{L}{2}|h_j|.$$

Furthermore, we note that the operator norm of a matrix $A \in \mathbb{R}^{m \times n}$ is given by[2]

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}| = \max_{1 \leq j \leq n} \|Ae_j\|_1$$

from which we obtain

$$\begin{aligned}\|\Delta_h f(x) - Df(x)\|_1 &= \max_{1 \leq j \leq n} \|\Delta_{h,j} f(x) - Df(x)e_j\|_1 \\ &\leq \max_{1 \leq j \leq n} \frac{L}{2}|h_j| = \frac{L}{2}\|h\|_\infty.\end{aligned}$$

□

This particular result comes in handy in the analysis of finite difference approximation errors in finite precision arithmetic (section 2.1.1) as well as in the convergence analysis of the Newton method where the Jacobians in each step are approximated using the forward difference formula (section 2.2). The inequality in (2.12) also gives us a general bound for the accuracy of approximating a directional derivative. Let $v = \sum_{j=1}^{n} \lambda_j e_j \in \mathbb{R}^n$ such that we seek to approximate $\partial_v f(x) = Df(x)v$ in $x \in G$ by the forward difference approximation viz. $\Delta_h f(x) \cdot v$. It then follows that

$$\begin{aligned}\|\partial_v f(x) - \Delta_h f(x) \cdot v\| &= \sum_{j=1}^{n} |\lambda_j| \|Df(x)e_j - \Delta_{h,j} f(x)\| \\ &\leq \frac{L}{2} \sum_{j=1}^{n} |\lambda_j||h_j| \\ &\leq \frac{L}{2}\|h\|_\infty \|v\|_1.\end{aligned}$$

[2] See Lemma A.1 in the Appendix.

2.1.1 The Effect of Finite Precision Arithmetic

Keeping to the one-dimensional case $f : G \to \mathbb{R}$, the relations (2.12) and (2.9) suggest ε should simply be chosen as small as possible in order to obtain the highest accuracy in approximating first order derivatives. However, if we account for finite precision arithmetic, the upper bound for the finite difference approximation exhibits a more complicated behaviour. Assume that f be bounded on G by some constant L_f, i.e. $|f(x)| \leq L_f$ for all $x \in G$. Let $\mathrm{comp}(f(x))$ denote the internal finite precision floating point representation of $f(x)$. Then the relative error

$$\frac{|\mathrm{comp}(f(x)) - f(x)|}{|f(x)|} \leq \mathbf{u}$$

is bounded by the *machine precision* $\mathbf{u}$. Since f is bounded by L_f, we conclude

$$|\mathrm{comp}(f(x)) - f(x)| \leq \mathbf{u} L_f$$

for all $x \in G$. This allows us to give an upper bound for the error of the computed finite difference quotient. Let $\varepsilon > 0$ such that $x + \varepsilon e_i \in G$, $i \in \{1, \ldots, n\}$. Then it holds

$$\begin{aligned}
&\left| \frac{\mathrm{comp}(f(x + \varepsilon e_i)) - \mathrm{comp}(f(x))}{\varepsilon} - \frac{f(x + \varepsilon e_i) - f(x)}{\varepsilon} \right| \\
&\leq \frac{1}{\varepsilon} \left(|\mathrm{comp}(f(x + \varepsilon e_i)) - f(x + \varepsilon e_i)| + |\mathrm{comp}(f(x)) - f(x)| \right) \\
&\leq \frac{1}{\varepsilon} 2\mathbf{u} L_f.
\end{aligned}$$

We obtain as an upper bound of the error of our approximation:

$$\begin{aligned}
& \left| \frac{\mathrm{comp}(f(x + \varepsilon e_i)) - \mathrm{comp}(f(x))}{\varepsilon} - \frac{\partial f}{\partial x_i}(x) \right| \\
\leq\ & \left| \frac{\mathrm{comp}(f(x + \varepsilon e_i)) - \mathrm{comp}(f(x))}{\varepsilon} - \frac{f(x + \varepsilon e_i) - f(x)}{\varepsilon} \right| + \left| \frac{f(x + \varepsilon e_i) - f(x)}{\varepsilon} - \frac{\partial f}{\partial x_i}(x) \right| \\
\leq\ & \frac{1}{\varepsilon} 2\mathbf{u} L_f + \frac{L}{2}\varepsilon =: \mathcal{E}(\varepsilon).
\end{aligned}$$

The function $\mathcal{E}$ is strictly convex, continuous and coercive and thus admits a unique minimizer ε^* which is exactly the point satisfying the necessary condition $\nabla \mathcal{E}(\varepsilon^*) = 0$. Hence, $\varepsilon^* = \sqrt{\frac{4 L_f \mathbf{u}}{L}}$. If we assume f to be "well-scaled", i.e. $\frac{L_f}{L} \approx 1$, choosing $\varepsilon^* = 1$ is going to cap the error at a fairly close to optimal value.

The same analysis can be conducted for the central difference formula (2.11). As before, we arrive at

$$\begin{aligned}
& \left| \frac{\mathrm{comp}(f(x + \varepsilon e_i)) - \mathrm{comp}(f(x - \varepsilon e_i))}{2\varepsilon} - \frac{\partial f}{\partial x_i}(x) \right| \\
= & \left| \frac{\mathrm{comp}(f(x + \varepsilon e_i)) - \mathrm{comp}(f(x - \varepsilon e_i))}{2\varepsilon} - \frac{f(x + \varepsilon e_i) - f(x - \varepsilon e_i)}{2\varepsilon} \right| \\
+ & \left| \frac{f(x + \varepsilon e_i) - f(x - \varepsilon e_i)}{2\varepsilon} - \frac{\partial f}{\partial x_i}(x) \right|.
\end{aligned}$$

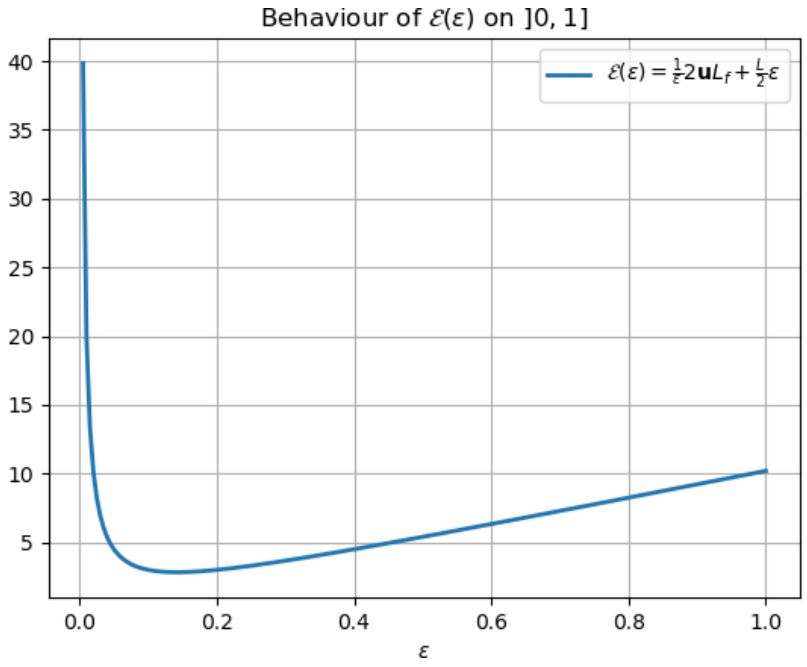


Figure 2.1: Exemplary plot of $\mathcal{E}$ for $L_f = 1$, $L = 20$, $\mathbf{u} = 0.1$ on $]0, 1]$

The first term can again be bounded from above by $\frac{1}{\varepsilon}\mathbf{u}L_f$ while the second term is in $o(\varepsilon^2)$ and can thus be bounded from above by $M\varepsilon^2$ for some constant $M > 0$. Hence, we obtain

$$\begin{aligned} & \left| \frac{\operatorname{comp}(f(x + \varepsilon e_i)) - \operatorname{comp}(f(x - \varepsilon e_i))}{2\varepsilon} - \frac{f(x + \varepsilon e_i) - f(x - \varepsilon e_i)}{2\varepsilon} \right| \\ + \quad & \left| \frac{f(x + \varepsilon e_i) - f(x - \varepsilon e_i)}{2\varepsilon} - \frac{\partial f}{\partial x_i}(x) \right| \\ \leq \quad & \frac{1}{\varepsilon}\mathbf{u}L_f + M\varepsilon^2 =: \mathcal{F}(\varepsilon). \end{aligned}$$

Again, $\mathcal{F}$ admits a unique minimizer ε^* satisfying $\nabla\mathcal{F}(\varepsilon^*) = 0$ resulting in $\varepsilon^* = \left(\frac{\mathbf{u}L_f}{2M}\right)^{\frac{1}{3}}$ with a corresponding error $\mathcal{F}(\varepsilon^*) = \mathbf{u}^{2/3}\left[L_f^{2/3}M^{1/3}(1 + 2^{1/3})\right] \in \mathcal{O}(\mathbf{u}^{2/3})$. Hence, we achieve a slight improvement in terms of accuracy by using using the central difference approximation at the expense of roughly doubling the computational effort which may be a sensible trade-off in some situations.

2.1.2 Colouring Techniques for Sparse Jacobians

Some optimization techniques may require an approximation of the whole Jacobian of a function in some point x. For instance, *Newton's method* for the approximation of the roots of a function $f : G \subseteq \mathbb{R}^n \to \mathbb{R}^m$, makes use of both the Jacobian of f and its inverse. If we use the forward difference formula to approximate $Df(x)$ using $\Delta_h f(x)$ and solve $\Delta_h f(x_k)d_{k+1} = b_k$, we require an approximation of the full Jacobian for some direct solution methods. Calculating the forward difference formula for every column of $\Delta_h f(x)$

can become very expensive – especially if f arises, for instance, from the discretization of a system of partial differential equations. However, if $Df(x)$ is known to be sparse, we might be able to exploit this by cleverly selecting perturbation vectors $d_1, \dots, d_p$ for some $p \in \{1, \dots, n\}$ such that for $1 \le j \le p$ and ε the difference formulae $\Delta_h f(x) d_j$ uniquely determine $\Delta_h f(x)$.

In other words, given $f : G \subseteq \mathbb{R}^n \to \mathbb{R}^m$, the goal is to determine the smallest number $p \in \mathbb{N}$ of vectors $d_1, \dots, d_p \in \mathbb{R}^n$ such that forward or central difference approximations of (2.10) and (2.11) uniquely determine the sparse Jacobian $Df(x)$ of f or more generally, that $Ad_1, \dots, Ad_p$ uniquely determine any given sparse $A \in \mathbb{R}^{m \times n}$.

The main contribution of [5] is to show that the solution to this problem is related to solving the colouring problem of a graph $G(A)$ determined by the sparsity pattern of A and that this problem is just as hard as the general graph colouring problem itself. It is thus implicitly assumed that the sparsity pattern of A is known. Several graph colouring algorithms running in time proportional to

$$\sigma := \sum_{i=1}^{n} \rho_i^2,$$

where ρ_i is the number of nonzero elements in the i-th row of A, are also investigated.

Specifying $Ad_1, \dots, Ad_p$ defines mp linear equations in the unknowns a_{ij}:

$$Ad_k = \begin{bmatrix} (Ad_k)_1 \\ \vdots \\ (Ad_k)_m \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{n} a_{1j} d_k^{(j)} \\ \vdots \\ \sum_{j=1}^{n} a_{mj} d_k^{(j)} \end{bmatrix}.$$

The resulting system of linear equations can also be written

$$\underbrace{\begin{array}{c} \\ m \\ \\ \\ \hline m \\ \\ \\ \hline \vdots \\ \hline m \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} n & n & \cdots & n \end{array} \\ \left[\begin{array}{cccc} d_1^T & 0 & \cdots & 0 \\ 0 & d_1^T & \cdots & 0 \\ \vdots & & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & d_1^T \\ \hline d_2^T & 0 & \cdots & 0 \\ 0 & d_2^T & \cdots & 0 \\ \vdots & & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & d_2^T \\ \hline \vdots & & & \vdots \\ \hline d_p^T & 0 & \cdots & 0 \\ 0 & d_p^T & \cdots & 0 \\ \vdots & & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & d_p^T \end{array}\right] \end{array}}_{=:\mathcal{E}} \begin{bmatrix} a_{11} \\ \vdots \\ a_{1n} \\ a_{21} \\ \vdots \\ a_{2n} \\ \vdots \\ a_{m1} \\ \vdots \\ a_{mn} \end{bmatrix} = \begin{bmatrix} (Ad_1)_1 \\ \vdots \\ (Ad_1)_m \\ (Ad_2)_1 \\ \vdots \\ (Ad_2)_m \\ \vdots \\ (Ad_p)_1 \\ \vdots \\ (Ad_p)_m \end{bmatrix}. \tag{2.14}$$

If we leave out the zeros already specified in the sparsity pattern of A, adjust the matrix $\mathcal{E}$ in (2.14) and denote by ρ_i the number of nonzero elements in row i of A such that only $\tau = \sum_{i=1}^{m} \rho_i$ unknowns remain, we see that A can only be uniquely determined if

$$mp \geq \tau \tag{2.15}$$

for otherwise, we have $\operatorname{rank}(\mathcal{E}) < \tau$ and we conclude from the rank-nullity theorem

$$\dim \ker(\mathcal{E}) = \tau - \operatorname{rank}(\mathcal{E}) > 0,$$

i.e. the kernel of $\mathcal{E}$ is nontrivial and therefore every solution of (2.14) will not be unique. Thus, $mp \geq \tau$ already provides a sharp bound for p which cannot be improved upon.

We say that $Ad_1, \ldots, Ad_p$ determine A *directly* if for every nonzero a_{ij} there exists some $d \in \{d_1, \ldots, d_p\}$ such that

$$a_{ij} d^{(j)} = (Ad)_i,$$

i.e. the sparsity pattern of A is exploited by cleverly arranging $d_1, \ldots, d_p$ such that all nonzero entries of A can be computed using at most one multiplication.

Thus, the problem we seek to solve is to obtain vectors $d_1, \ldots, d_p$ such that $Ad_1, \ldots, Ad_p$ determine A directly with p being as small as possible. To this end, it is helpful to realize that for an $m \times n$ matrix to be determined directly, a group of columns is determined by only one evaluation of Ad if no two columns in said group have a nonzero entry in the same row. For this purpose, let $a_1, \ldots, a_n$ denote the columns of A and let $G := \{a_j \mid j \in C\}$ denote a group of columns with some index set $C \subset \{1, \ldots, n\}$ such that no two columns in this group share a row where they both have nonzero entries. Setting for $\lambda_j \in \mathbb{R}$ and

$$d = \sum_{j \in C} \lambda_j e_j, \ (e_j)_i = \delta_{ij}, \tag{2.16}$$

such that

$$d^{(j)} = \begin{cases} \lambda_j & , \text{ if } j \in C \\ 0 & , \text{ else} \end{cases},$$

it follows

$$Ad = \sum_{j \in C} \lambda_j a_j,$$

and since no two columns of G share nonzero elements within the same row, we have for every $i \in \{1, \ldots, m\}$

$$(Ad)_i = \sum_{j \in C} \lambda_j a_{ij} = \lambda_j a_{ij} \tag{2.17}$$

which means that the columns of G are directly determined by Ad. Motivated by this, we call a division of the columns of A into mutually disjoint groups $C_1, \ldots, C_p$ *consistent* with the direct determination of A if no two columns of each individual group share a nonzero element in the same row and if each column of A belongs to one and only one group. This leads us to the core problem we need to solve in order to determine A directly with an approximation of the smallest number of columns.

Partition Problem	
Input:	An $m \times n$ matrix $A \in \mathbb{R}^{m \times n}$.
Task:	Obtain a consistent partition of the columns of A with the smallest number p of groups, i.e. find mutually disjoint sets $C_1, \ldots, C_p$ of columns of A such that no two columns within each group have a nonzero entry within the same row.

Note that both problems are not necessarily equivalent, i.e. the solution p and corresponding $d_1, \ldots, d_p$ of the partition problem is not necessarily the optimal solution to the original problem of finding the smallest number of vectors $d_1, \ldots, d_p$ to determine A directly. For instance, if we denote by $\gamma(A)$ the number of groups in an optimal partition of A, we can see that for any matrix

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

for some matrices A_1, A_2 where $\gamma(A) > \gamma(A_1) + \gamma(A_2)$, taking the respectively corresponding sets of vectors $d_1, \ldots, d_{\gamma(A_1)}$, and $b_1, \ldots, b_{\gamma(A_2)}$ we have

$$(Ad_j)_i = d_j^{(i)}(A_1)_{ij}$$

and

$$(Ab_j)_i = b_j^{(i)}(A_2)_{ij}$$

directly determining A using the $\gamma(A_1) + \gamma(A_2)$ vectors $d_1, \ldots, d_{\gamma(A_1)}$, $b_1, \ldots, b_{\gamma(A_2)}$. The existence of such an example is pointed out in [5]. However, instead of studying the general problem of determining A with the smallest amount of columns p, Coleman and Moré [5] proceed to show that direct determination of A based on the partition problem is close to optimal for some randomly generated matrices as well as for some 30 matrices arising from finite element discretization schemes by comparing the determined number of columns p with a lower bound similar to the one derived in (2.15).

But how can we determine A directly using the consistent column partition approach? We do this by reducing it to a *graph colouring* problem. Recall that for a graph $G = (V, E)$ a p-colouring is a function

$$\Phi : V \to \{1, \ldots, p\}$$

such that $\Phi(u) \neq \Phi(v)$ for any two $\{u, v\} \in E$. Naturally, a p-colouring induces a partition of V with the components $C_1, \ldots, C_p$ where

$$C_i = \{v \in V \,|\, \Phi(v) = i\}.$$

Let us then define $G(A) = (V, E)$ by virtue of

$$\begin{aligned} V &:= \{1, \ldots, n\} \\ E &:= \{\{i, j\} \in \mathcal{P}(V) \,|\, \exists k \in \{1, \ldots, m\} : a_{ki} \neq 0 \wedge a_{kj} \neq 0\}, \end{aligned}$$

i.e. two vertices are connected if and only if their respective columns of A both have nonzero entries in one row. This is known as the *intersection graph* or the *column incidence graph* of A. Thus, for a given colouring inducing a partition $C_1, \ldots, C_p$, setting $d_1, \ldots, d_p$ according to (2.16) yields (2.17) directly determining A which means that a colouring of $G(A)$ induces a consistent partition of the columns of A and vice versa.

Theorem 2.4 (Theorem 2.1 in [5]). *Φ is a colouring of $G(A)$ if and only if Φ induces a consistent partitioning of the columns of A.*

The partition problem is thus equivalent to the following problem.

GRAPH COLOURING PROBLEM	
Input:	A graph $G = (V, E)$.
Task:	Find a p and a function $\Phi : V \to \{1, \dots, p\}$ such that $\Phi(u) \neq \Phi(v)$ whenever $\{u, v\} \in E$.

The connection of the partition problem to the graph colouring problem is made obvious now. It can be a helpful concept seeing as the intersection graph of A is invariant under row and column permutations and that there exists extensive literature treating the graph colouring problem. For some types of Jacobians, the chromatic number and efficient algorithms can be designed to generate a colouring – and thus a partitioning – to directly determine the entire Jacobian. Thus, it can be of great benefit to analyze the colouring problem for particular classes of Jacobians or Hessians which we seek to approximate using finite differences or even other techniques (such as algorithmic differentiation, see section 3). Additionally, for sufficiently smooth functions, Hessians of scalar functions are symmetric and allow for certain modifications of the reduction to the graph colouring problem. This is described in more detail in [6]. For instance, it is easily recognized that a tridiagonal matrix A induces an intersection graph $G(A)$ which has a chromatic number $\chi(G(A))$ of 3 and therefore, A is directly determined by no more than three evaluations of Ad. Coleman and Moré [5] go on to prove that the graph colouring problem for intersection graphs $G(A)$ of matrices $A \in \mathbb{R}^{m \times n}$ is NP-complete, but present four different graph colouring algorithms in order to approximate solutions to the partition problem.

As an example, consider the vector-valued function $r : G \subseteq \mathbb{R}^n \to \mathbb{R}^m$, our goal is to approximate the Jacobian $Dr(x)$ of r:

$$Dr(x) = \left[\frac{\partial r_i}{\partial x_j}(x)\right]_{i=1,\dots,m}^{j=1,\dots,n} = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{2.18}$$

where r_j for $j = 1, \dots, m$ are the component functions of r.

Suppose r is continuously differentiable and Dr is Lipschitz-continuous on the domain G (for instance, if $Dr : G \subseteq \mathbb{R}^n \to L(\mathbb{R}^n, \mathbb{R}^m)$ is found to be continuously differentiable on some compact, convex subset $K \supset G \subseteq \mathbb{R}^n$). Then the forward difference approximation for $p \in \mathbb{R}^n$ such that $x + p \in G$

$$Dr(x)p \approx \frac{r(x + \varepsilon p) - r(x)}{\varepsilon}$$

approximates $Dr(x)p$ with an accuracy up to $\mathcal{O}(\varepsilon)$ (see subsection 2.1).

Let $r(x)$ be given by

$$r(x) = \begin{bmatrix} 2(x_2^3 - x_1^2) \\ 3(x_2^3 - x_1^2) - 2(x_3^3 - x_2^2) \\ 3(x_3^3 - x_2^2) - 2(x_4^3 - x_3^2) \\ \vdots \\ 2(x_n^3 - x_{n-1}^2) \end{bmatrix}. \tag{2.19}$$

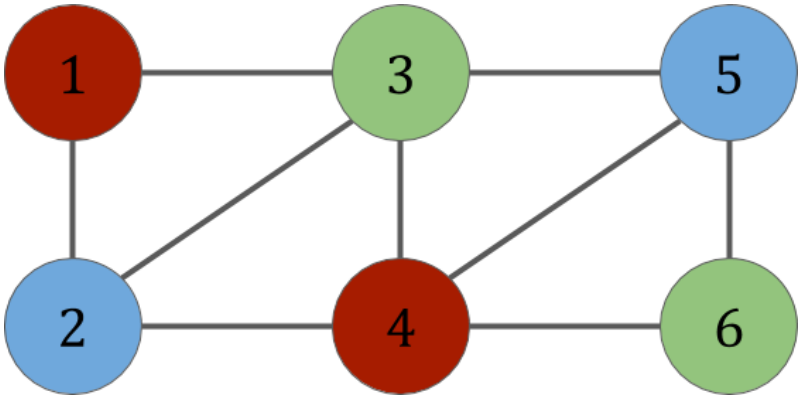


Figure 2.2: Exemplary colouring of the sparsity pattern matrix intersection graph $G(Dr(x))$.

Seeing as each component of r depends only on two or three components of x, the Jacobian has the following structure in the case of $n = 6$:

$$\begin{bmatrix} \times & \times & & & & \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}, \tag{2.20}$$

where each cross represents a nonzero element.

The corresponding intersection graph of $Dr(x)$ and a valid colouring can be seen in Figure 2.2. We see that the colouring induces a column partition into three groups

$$\begin{aligned} C_1 &= \{1, 4, 7, \ldots\} \\ C_2 &= \{2, 5, 8, \ldots\} \\ C_3 &= \{3, 6, 9, \ldots\} \end{aligned}$$

and thus

$$\begin{aligned} d_1 &= \varepsilon(e_1 + e_4 + e_7 + \ldots) \\ d_2 &= \varepsilon(e_2 + e_5 + e_8 + \ldots) \\ d_3 &= \varepsilon(e_3 + e_6 + e_9 + \ldots) \end{aligned}$$

define a set of vectors which directly determine $Dr(x)$ uniquely.

2.2 Application: Newton's Method with Finite Difference Approximations

Conventionally, Newton's method for the approximation of roots of a function $f : G \subseteq \mathbb{R}^n \to \mathbb{R}^n$ for G open is proved under the assumption that $Df(x)$ for $x \in G$ can be determined exactly. In the case that $Df(x)$ is approximated with the forward difference method such that instead of the standard Newton iteration

$$x_{k+1} = x_k - Df(x_k)^{-1} f(x_k)$$

for $k \in \{0, 1, 2, \ldots\}$, the approximated Newton iteration

$$x_{k+1} = x_k - \Delta_h f(x_k)^{-1} f(x_k)$$

is computed, it is still possible to prove local convergence for this method albeit only linear convergence in the 1-norm instead of quadratic convergence. We shall examine one way of proving this loosely following the arguments made in [10].

We assume that

(i) f is continuously differentiable,

(ii) there exists some $x^* \in G$ such that $f(x^*) = 0$,

(iii) Df is Lipschitz-continuous with a Lipschitz constant $L > 0$ in some open ball $U_\varepsilon(x^*)$ of radius $\varepsilon > 0$ around x^*,

(iv) $Df(x^*)$ is nonsingular, i.e. $Df(x^*)^{-1}$ exists.

We set $\frac{\beta}{2} = \|Df(x^*)^{-1}\|_1$. The essential pieces of the proof for the convergence of the Newton method are put together by iterated continuity arguments.

Since $\mathbf{det} : \mathbb{R}^{n\times n} \to \mathbb{R}$ is a continuous mapping[3] there exists some open neighbourhood U_1 of x^* such that $Df(x)^{-1}$ also exists for any $x \in U_1$. But because the mapping[4] $\mathbf{inv} : \mathrm{GL}(n, \mathbb{R}) \to \mathrm{GL}(n, \mathbb{R})$, $A \mapsto A^{-1}$ is also continuous, we see that there exists some $\varepsilon_1 > 0$ such that $\|A^{-1} - Df(x^*)^{-1}\|_1 \leq \frac{\beta}{2}$ whenever $\|A - Df(x^*)\|_1 < \varepsilon_1$ for nonsingular $A \in \mathbb{R}^{n\times n}$. By the continuity of $\mathbf{det}$ again, we find some neighbourhood $U_2 \subseteq U_1$ of x^* such that $\|Df(x) - Df(x^*)\|_1 < \varepsilon_1$ for all $x \in U_2$. This shows that $\|Df(x)^{-1} - Df(x^*)^{-1}\|_1 \leq \frac{\beta}{2}$ whenever $x \in U_2$. We deduce that

$$\|Df(x)^{-1}\|_1 \leq \|Df(x)^{-1} - Df(x^*)^{-1}\|_1 + \|Df(x^*)^{-1}\|_1 \leq \frac{\beta}{2} + \frac{\beta}{2} = \beta. \tag{2.21}$$

Now Theorem 2.3 states that for a given pertubation stepsize vector $h \in \mathbb{R}^n$, we have

$$\|\Delta_h f(x) - Df(x)\|_1 \leq \frac{L}{2}\|h\|_\infty. \tag{2.22}$$

Since we can select h in such a way that $\|h\|_\infty$ becomes arbitrarily small, we can argue with the continuity of both $\mathbf{det}$ and $\mathbf{inv}$ as above to produce a neighbourhood $U_3 \subseteq U_2 \subseteq U_1$ such that for all $x \in U_3$ holds $\|\Delta_h f(x)^{-1} - Df(x)^{-1}\|_1 \leq \beta$. It follows that

$$\|\Delta_h f(x)^{-1}\|_1 \leq \|\Delta_h f(x)^{-1} - Df(x)^{-1}\|_1 + \|Df(x)^{-1}\|_1 \leq \beta + \beta = 2\beta \tag{2.23}$$

for all $x \in U_4$. Thus, we select, without loss of generality, $\varepsilon > 0$ small enough such that conditions (2.21) and (2.23) both hold for $x \in U_\varepsilon(x^*)$. With these ingredients, we are now able to formulate the following theorem.

[3] This can be seen by recognizing that the Leibniz formula for the computation of the determinant is a sum of compositions of continuous mappings.

[4] Since $A^{-1} = \frac{1}{\det(A)}\mathbf{Cof}(A)^T$ which is a continuous mapping where $\mathbf{Cof}(A)$ is the *cofactor matrix* of A, we conclude that $\mathbf{inv}$ is continuous. See also [3, p.4], and [4, pp.457–458] where $\mathbf{inv}$ is even shown to be differentiable.

Theorem 2.5. *Let $f : G \subseteq \mathbb{R}^n \to \mathbb{R}^n$ with G open such that f and Df fulfill properties (i) to (iv) and consequently (2.21) to (2.23). Then there exists some $\varepsilon > 0$ and an $\|h\|_\infty =: \eta > 0$ such that for any $x_0 \in U_\varepsilon(x^*)$ the approximated Newton iteration*

$$x_{k+1} = x_k - \Delta_h f(x_k)^{-1} f(x_k) \tag{2.24}$$

is well-defined and converges linearly to x^*.

Proof.

We saw in the preceding remarks that $\|h\|_\infty = \eta$ and $\varepsilon > 0$ can be chosen sufficiently small such that for $x_0 \in U_\varepsilon(x^*)$ the expression (2.24) is well-defined. We now show that if $x_k \in U_\varepsilon(x^*)$, so is x_{k+1} and that $\|x_{k+1} - x^*\|_1 \leq c\|x_k - x^*\|_1$ for $c < 1$ which completes the proof.

Let x_{k+1} be the result of (2.24) where $x_k \in U_\varepsilon(x^*)$. Then it follows:

$$\begin{aligned} x_{k+1} - x^* &= x_k - \Delta_h f(x_k)^{-1} f(x_k) - x^* \\ &= \Delta_h f(x_k)^{-1} \overbrace{f(x^*)}^{=0} - \Delta_h f(x_k)^{-1} f(x_k) - (x^* - x_k) \\ &= \Delta_h f(x_k)^{-1} \left[f(x^*) - f(x_k) - \Delta_h f(x_k)(x^* - x_k) \right]. \end{aligned}$$

Taking the norm on both sides, it follows that

$$\begin{aligned} \|x_{k+1} - x^*\|_1 &\leq \|\Delta_h f(x_k)^{-1}\|_1 \|f(x^*) - f(x_k) - \Delta_h f(x_k)(x^* - x_k)\|_1 \\ &\leq 2\beta \|f(x^*) - f(x_k) - \Delta_h f(x_k)(x^* - x_k)\|_1 \qquad \text{(Application of (2.23))}. \end{aligned}$$

In order to estimate $\|f(x^*) - f(x_k) - \Delta_h f(x_k)(x^* - x_k)\|_1$ further, we aim to apply both (2.4) from Lemma 2.2 as well as (2.22) from Theorem 2.3.

$$\begin{aligned} \|f(x^*) - f(x_k) - \Delta_h f(x_k)(x^* - x_k)\|_1 &= \big\| f(x^*) - f(x_k) - \Delta_h f(x_k)(x^* - x_k) \\ &\quad - [f(x_k) + Df(x_k)(x^* - x_k)] \\ &\quad + [f(x_k) + Df(x_k)(x^* - x_k)] \big\|_1 \\ &= \big\| [f(x^*) - f(x_k) - Df(x_k)(x^* - x_k)] \\ &\quad + [Df(x_k) - \Delta_h f(x_k)](x^* - x_k) \big\|_1 \\ &\leq \|f(x^* - f(x_k) - Df(x_k)(x^* - x_k))\|_1 \\ &\quad + \|Df(x_k) - \Delta_h f(x_k)\|_1 \|x^* - x_k\|_1 \\ \text{(Application of (2.4) and (2.22))} &\leq \frac{L}{2}\|x^* - x_k\|_1^2 + \frac{L}{2}\eta\|x^* - x_k\|_1. \end{aligned}$$

Plugging this into the expression for $\|x_{k+1} - x^*\|_1$, we obtain

$$\begin{aligned} \|x_{k+1} - x^*\|_1 &\leq 2\beta \left(\frac{L}{2}\|x^* - x_k\|_1 + \frac{L}{2}\eta\|x^* - x_k\|_1 \right) \\ &= \beta L \|x^* - x_k\|_1 (\|x^* - x_k\|_1 + \eta). \end{aligned}$$

Now, if $\eta > 0$ is chosen such that $\eta < \frac{1}{2\beta L}$, then selecting $0 < \varepsilon < \frac{1}{2\beta L} - \eta$ yields

$$\|x_{k+1} - x^*\|_1 \leq \frac{1}{2}\|x_k - x^*\|_1$$

which proves the theorem. □

This shows that at least linear convergence of the Newton method for any initial value sufficiently close to x^* is preserved even if the Jacobians $Df(x_k)$ are approximated by $\Delta_h f(x_k)$ using the forward difference formula.

2.3 Approximation of Second Order Derivatives

The concepts of the previous sections explain how to approximate first order derivatives – and thus gradients and Jacobians – but can be fairly easily extended to compute higher order derivatives such as the Hessian $D^2 f(x)$ of a scalar function $f : G \subseteq \mathbb{R}^n \to \mathbb{R}$.

For instance, in the case of a scalar function, the forward difference or central difference approximations can be applied to $\nabla f : G \subseteq \mathbb{R}^n \to \mathbb{R}^n$ yielding

$$Df^2(x)p \approx \frac{\nabla f(x + \varepsilon p) - \nabla f(x)}{\varepsilon} \tag{2.25}$$

which converges linearly to $D^2 f(x)p$ as $\varepsilon \to 0$. However, applying this to an already approximated value of $\nabla f(x)$ introduces another layer of approximation errors and might result in non-symmetric approximations of the Hessian even if $f \in \mathcal{C}^2(G)$ in which case $D^2 f(x)$ would be symmetric according to Schwarz's lemma [4, p.500]. Let $D_h^2 f(x)$ denote the approximation according to (2.25) (obtained column-wise for $p = e_j$, $j \in \{1, \ldots, n\}$). In that case, one might set $D^2 f(x) \approx \frac{1}{2}(D_h^2 f(x) + D_h^2 f(x)^T) =: P(D_h^2 f(x))$ which is symmetric. This definition also makes sense geometrically since $P(D_h^2 f(x))$ is the projection of $D_h^2 f(x)$ onto the subspace $\mathrm{Sym}(n, \mathbb{R})$ of symmetric, real $n \times n$ matrices with respect to the Frobenius norm $\|A\|_F = \sqrt{\langle A, A\rangle_F} = \sqrt{\mathrm{tr}(A^T A)}$, $A \in \mathbb{R}^{n\times n}$. This follows from the projection theorem [4, pp.183–184] given that any finite-dimensional subspace of a vector space over a complete field is itself complete and closed. Indeed, for any $Z \in \mathrm{Sym}(n, \mathbb{R})$, and general $A \in \mathbb{R}^{n\times n}$, we have

$$\begin{aligned}
\langle P(A) - A, Z\rangle_F &= \left\langle \frac{1}{2}(A + A^T) - A, Z \right\rangle_F \\
&= \left\langle \frac{1}{2}(A^T - A), Z \right\rangle_F \\
&= \frac{1}{2}\left(\langle A^T, Z\rangle_F - \langle A, Z\rangle_F\right) \\
&= \frac{1}{2}\left(\mathrm{tr}(A^T Z) - \mathrm{tr}(AZ)\right) = 0
\end{aligned}$$

since $\mathrm{tr}(AZ) = \mathrm{tr}(ZA) = \mathrm{tr}(A^T Z)$ which is easily verified.

Nonetheless, we can derive similar direct approximation formulae as in the previous section by invoking Taylor's theorem. In the case that $f \in \mathcal{C}^3(G)$, setting $p = \varepsilon e_i$, $p = \varepsilon e_j$,

and $p = \varepsilon(e_i + e_j)$, we obtain:

$$\begin{aligned}
f(x + \varepsilon e_i) &= f(x) + \varepsilon \frac{\partial f}{\partial x_i}(x) + \frac{1}{2}\varepsilon^2 \frac{\partial^2 f}{\partial x_i^2}(x) + o(\varepsilon^3) \\
f(x + \varepsilon e_j) &= f(x) + \varepsilon \frac{\partial f}{\partial x_j}(x) + \frac{1}{2}\varepsilon^2 \frac{\partial^2 f}{\partial x_j^2}(x) + o(\varepsilon^3) \\
f(x + \varepsilon(e_i + e_j)) &= f(x) + \varepsilon \left(\frac{\partial f}{\partial x_i}(x) + \frac{\partial f}{\partial x_j}(x) \right) \\
&\quad + \frac{1}{2}\varepsilon^2 \left(\frac{\partial^2 f}{\partial x_i^2}(x) + 2\frac{\partial^2 f}{\partial x_i \partial x_j}(x) + \frac{\partial^2 f}{\partial x_j^2}(x) \right) + o(\varepsilon^3). \quad \text{(Schwarz's lemma)}
\end{aligned}$$

Subtracting the first and second equations from the third yields

$$f(x + \varepsilon(e_i + e_j)) - f(x + \varepsilon e_i) - f(x + \varepsilon e_j) = -f(x) + \varepsilon^2 \frac{\partial^2 f}{\partial x_i \partial x_j}(x) + o(\varepsilon^3),$$

whence

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) = \frac{f(x + \varepsilon(e_i + e_j)) - f(x + \varepsilon e_i) - f(x + \varepsilon e_j) + f(x)}{\varepsilon^2} + o(\varepsilon). \tag{2.26}$$

This necessitates the evaluation of f in all $\frac{n(n+1)}{2}$ combinations of $x + \varepsilon(e_i + e_j)$, $i, j \in \{1, \dots, n\}$ as well as in all $x + \varepsilon e_i$ and in x. If the Hessian is known to be sparse, we can of course apply techniques similar to the ones used in section 2.1.2 in order to skip the evaluation whenever we know $\frac{\partial^2 f}{\partial x_i \partial x_j}(x + p)$ to be zero.

2.4 Application: Numerical Solution of Two-Point Boundary Value Problems

One application of approximating derivatives using finite differences is the so-called *finite difference method* for the numerical solution of ordinary or partial differential equations. Consider, for instance, the following one-dimensional boundary value problem on $\Omega =]\alpha, \beta[\subseteq \mathbb{R}$

$$\begin{cases} \text{Find } u \in \mathcal{C}^2(\Omega) \cap \mathcal{C}(\overline{\Omega}) \text{ such that} \\ L[u](x) := -a(x)u''(x) + b(x)u'(x) + c(x)u(x) = f(x) \quad \text{in } \Omega \\ u(\alpha) = u_\alpha,\ u(\beta) = u_\beta \end{cases} \tag{2.27}$$

for some right hand side and coefficient functions $f, a, b, c \in \mathcal{C}(\overline{\Omega})$. Imposing the conditions $a > 0, c \geq 0$ on Ω ensures the existence and uniqueness of the solution, see e.g. [11, p.9]. The basic idea is to define an equidistant grid

$$\begin{aligned}
\Omega_h &= \{x_j := \alpha + jh \,|\, j \in \{1, \dots, N\}\} \\
\overline{\Omega}_h &= \{x_j := \alpha + jh \,|\, j \in \{0, \dots, N+1\}\}
\end{aligned}$$

where $h = \frac{\beta - \alpha}{N+1}$ for some $N \in \mathbb{N}$ and consider a discretized version of (2.27) on a space of *grid functions*

$$\begin{aligned}
G_h &= \{(U_j)_{j=1}^N =: U(x_j) \,|\, U_j \in \mathbb{R}\} = \mathbb{R}^N \\
\overline{G}_h &= \{(U_j)_{j=0}^{N+1} =: U(x_j) \,|\, U_j \in \mathbb{R}\} = \mathbb{R}^{N+2}
\end{aligned}$$

by replacing the derivatives with forward and central difference approximations:

$$\begin{aligned} L_h[U](x_j) := &- a(x_j)\frac{U(x_{j+1}) - 2U(x_j) + U(x_{j-1})}{h^2} \\ &+ b(x_j)\frac{U(x_{j+1}) - U(x_{j-1})}{2h} \\ &+ c(x_j)U(x_j) = f(x_j) \end{aligned}$$

such that the discretized problem is given by

$$\begin{cases} \text{Find } U \in \overline{G}_h \text{ such that} \\ L_h[U](x_j) = f(x_j) & \text{in } \Omega_h\,. \\ U(x_0) = U(\alpha) = u_\alpha,\ U(x_{N+1}) = U(\beta) = u_\beta \end{cases} \tag{2.28}$$

We can demonstrate the existence of a unique solution to this problem by, for instance, proving that the linear system matrix equivalent to (2.28) is nonsingular. A similar strategy is pursued in [11, ch.3]. We construct the system matrix by multiplying both sides of (2.28) with $\frac{h^2}{2}$ and solving the resulting system:

$$\begin{aligned} \frac{h^2}{2}L_h[U](x_j) &= -\frac{1}{2}a(x_j)(U_{j+1} - 2U_j + U_{j-1}) + \frac{h}{4}b(x_j)(U_{j+1} - U_{j-1}) + \frac{h^2}{2}c(x_j)U_j \qquad (2.29) \\ &= -\frac{1}{2}\left[a(x_j) + \frac{h}{2}b(x_j)\right]U_{j-1} + \left[a(x_j) + \frac{h^2}{2}c(x_j)\right]U_j - \frac{1}{2}\left[a(x_j) - \frac{h}{2}b(x_j)\right]U_{j+1} \\ &= a_j U_{j-1} + b_j U_j + c_j U_{j+1} \end{aligned}$$

for $j = 1, \ldots, N$. With the boundary values $U_0 = u_\alpha$ for $j = 1$ and $U_{N+1} = u_\beta$ for $j = N$, this defines a linear system $L_h U = g$ with a modified right-hand side:

$$L_h U = \begin{bmatrix} b_1 & c_1 & 0 & \cdot & \cdot & \cdot & 0 \\ a_2 & b_2 & c_2 & & & & \\ & \cdot & \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & \cdot & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot & \cdot & \cdot \\ \cdot & & & & a_{N-1} & b_{N-1} & c_{N-1} \\ 0 & \cdot & \cdot & \cdot & 0 & a_N & b_N \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \cdot \\ \cdot \\ \cdot \\ U_N \end{bmatrix} = \frac{h^2}{2} \begin{bmatrix} f(x_1) \\ f(x_2) \\ \cdot \\ \cdot \\ \cdot \\ f(x_{N-1}) \\ f(x_N) \end{bmatrix} - \begin{bmatrix} a_1 u_\alpha \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ c_N u_\beta \end{bmatrix} =: g. \tag{2.30}$$

We prove that there exists an LU-decomposition $L_h = LR$ of L_h with L being a nonsingular lower triangular matrix and R being a nonsingular upper triangular matrix defined by the ansatz:

$$L = \begin{bmatrix} \beta_1 & 0 & \cdot & \cdot & 0 \\ a_2 & \beta_2 & & & \cdot \\ 0 & \cdot & \cdot & & \cdot \\ \cdot & & \cdot & \cdot & 0 \\ 0 & \cdot & 0 & a_N & \beta_N \end{bmatrix}, \quad R = \begin{bmatrix} 1 & \gamma_1 & 0 & \cdot & 0 \\ 0 & 1 & \gamma_2 & & \cdot \\ \cdot & & \cdot & \cdot & 0 \\ \cdot & & & \cdot & \gamma_{N-1} \\ 0 & \cdot & \cdot & 0 & 1 \end{bmatrix}. \tag{2.31}$$

The requirement $L_h = LR$ implies that

$$\begin{aligned} &\beta_1 = b_1, \quad \gamma_1 = \frac{c_1}{\beta_1} \\ &\beta_j = b_j - a_j\gamma_{j-1}, && 2 \le j \le N \\ &\gamma_j = \frac{c_j}{\beta_j}, && 2 \le j \le N-1. \end{aligned} \tag{2.32}$$

Setting, for the sake of conciseness, in L_h the coefficients $a_1 = c_N = 0$ – in distinction to the corresponding coefficients from (2.29) which were included in the right-hand side (2.30) – we can derive a sufficient condition for the existence of the proposed LU-decomposition as well as for the nonsingularity of L_h.

Theorem 2.6. *Let the elements of* L_h *in* (2.30) *satsify*

$$|b_j| > |a_j| + |c_j|, \quad 1 \le j \le N, \quad a_1 = c_N = 0. \tag{2.33}$$

Then L_h *is nonsingular and the proposed LU-decomposition with the ansatz in* (2.31) *is well-defined. In addition, the quantities* β_j *and* γ_j *are bounded by*

$$|\gamma_j| < 1 \tag{2.34}$$

$$0 < |b_j| - |a_j| \le |\beta_j| \le |b_j| + |a_j|. \tag{2.35}$$

Proof.

To prove the nonsingularity, we show that the decomposition is well-defined and that all $\beta_j \neq 0$ for in that case we have

$$\det(L_h) = \det(LR) = \det(L)\det(R) = \prod_{j=1}^{N} \beta_j \neq 0.$$

For $j = 1$ assumption (2.33) gives $|b_1| > |c_1|$ which implies $\beta = b_1 \neq 0$ and $\gamma_1 = \frac{c_1}{\beta_1} = \frac{c_1}{b_1}$ is well-defined and both coefficients satisfy (2.34), (2.35).

Assume now that $|\gamma_{j-1}| < 1$ for some $1 \le j \le N$. Then

$$|\beta_j| = |b_j - a_j\gamma_{j-1}| \le |b_j| + |a_j||\gamma_{j-1}| \le |b_j| + |a_j|.$$

In addition, by the reverse triangle inequality, we obtain

$$|\beta_j| = |b_j - a_j\gamma_{j-1}| \ge ||b_j| - |a_j||\gamma_{j-1}||.$$

But due to $|\gamma_{j-1}| < 1$, we have $|b_j| - |a_j||\gamma_{j-1}| \ge |b_j| - |a_j| > |c_j| \ge 0$ by (2.33). Thus follows (2.35) which also implies $\beta_j \neq 0$. Hence, γ_j is well-defined and using the same estimates, we see that

$$|\gamma_j| = \frac{|c_j|}{|b_j - a_j\gamma_{j-1}|} \le \frac{|c_j|}{|b_j| - |a_j||\gamma_{j-1}|} \le \frac{|c_j|}{|b_j| - |a_j|} < 1$$

due to (2.33). This proves that L_h is nonsingular and an LU-decomposition is given by (2.31). □

It follows that there exists a unique solution of (2.28) if we can find some $h > 0$ such that the coefficients of L_h satisfy (2.33).

Corollary 2.7. *Let* $a_0, B > 0$, $b \in \mathbb{R}$ *such that* $b \le b(x) \le B$, $0 < a_0 \le a(x)$, *and* $c(x) > 0$ *for all* $x \in \Omega$. *If* $h > 0$ *is chosen such that it satisfies*

$$h < \frac{a_0}{b} \tag{2.36}$$

then the coefficients of (2.30) *satisfy* $|b_j| > |a_j| + |c_j|$ *implying that* (2.28) *has a unique solution.*

Proof.

We show that the proposed choice of h implies that for $1 \le j \le N$

$$a(x_j) \pm \frac{h}{2} b(x_j) > 0 \tag{2.37}$$

for in that case, we have

$$|b_j| = \left| a(x_j) + \frac{h^2}{2} c(x_j) \right| \ge a(x_j) + \frac{h^2}{2} \underbrace{c(x_j)}_{>0} > a(x_j)$$

and

$$\begin{aligned} |b_j| = b_j &> a(x_j) \\ &= \frac{1}{2} a(x_j) + \frac{h}{4} b(x_j) + \frac{1}{2} a(x_j) - \frac{h}{4} b(x_j) \\ &= \frac{1}{2} \left[a(x_j) + \frac{h}{2} b(x_j) \right] + \frac{1}{2} \left[a(x_j) - \frac{h}{2} b(x_j) \right] \\ &= a_j + c_j = |a_j| + |c_j|. \end{aligned}$$

Hence, it only remains to show (2.37). We see that whenever $b(x_j) \ge 0$

$$a_j = a(x_j) + \frac{h}{2} b(x_j) \ge a_0 > 0.$$

If $b(x_j) < 0$, implying $b < 0$, we choose h appropriately and since $-b < B$, we have

$$\frac{h}{2} b(x_j) > \frac{a_0}{2B} b(x_j) > \frac{a_0}{-2b} b(x_j) \ge -\frac{a_0}{2}.$$

Combining this with a_j we get

$$a_j = a(x_j) + \frac{h}{2} b(x_j) \ge a_0 - \frac{a_0}{2} = \frac{a_0}{2} > 0.$$

A similar argument also yields $c_j > 0$ proving the claim. □

Applying Taylor's theorem as in the previous sections, we can even derive a discretization error estimate. For any smooth function $u \in \mathcal{C}^2(\Omega) \cap \mathcal{C}(\overline{\Omega})$, we define the *local truncation error*

$$\tau_j[u] = L_h[R_h[u]](x_j) - L[u](x_j)$$

where $R_h[u](x_j) = u(x_j)$, i.e. $R_h[u] \in \overline{G}_h$. Expanding u around every x_j in the directions $\pm h$ up to the third and fourth orders respectively, we find

$$\tau_j[u] = -\frac{h^2}{12}\left[u^{(4)}(\eta_j^{(1)}) + u^{(4)}(\eta_j^{(2)}) - 2b(x_j)\left(u^{(3)}(\xi_j^{(1)}) - u^{(3)}(\xi_j^{(2)})\right)\right] \tag{2.38}$$

for $\eta_j^{(1)}$, $\xi_j^{(1)} \in [x_j, x_{j+1}]$, and $\eta_j^{(2)}$, $\xi_j^{(2)} \in [x_{j-1}, x_j]$.

This shows that the discretization is *consistent with* L, i.e. $\tau_j[u] \to 0$ as $h \to 0$. It can also be shown (see [11, pp.76–77]) that there exists some $h > 0$ and a constant $M > 0$ independent of h such that for all $1 \le j \le N$

$$|U_j| \le M\left(\max\{|U_0|, |U_{N+1}|\} + \max_{1\le j\le N} |L_h[U](x_j)|\right)$$

which is called a *stability estimate* since it shows that solutions to the discretized problem (2.28) continuously depend on the boundary values and the maximum norm of the right-hand side f. Together with (2.38), we conclude that for any solution u of the continuous problem (2.27) which has continuous fourth derivatives and corresponding solution U of (2.28), the *approximation error* is bounded by

$$|R_h[u](x_j) - U(x_j)| \le M\frac{h^2}{12}\left(2\max_{x\in[\alpha,\beta]}\left|\frac{d^4}{dx^4}u(x)\right| + 4\max_{x\in[\alpha,\beta]}|b(x)|\cdot\max_{x\in[\alpha,\beta]}\left|\frac{d^3}{dx^3}u(x)\right|\right)$$

for all $1 \le j \le N$ (cf. [11, p.77]). This means that the finite difference approximation of (2.27) has a quadratic convergence rate for solutions u with bounded third and fourth derivatives.

3 Algorithmic Differentiation

Algorithmic differentiation (by some authors also called automatic differentiation) refers to a set of techniques making use of the computational representation of functions in order to produce analytic (exact) values for the sought-after derivatives. While some techniques do this by altering the function code directly, others keep a record of the computations performed in order to evaluate the function at some specific point x and use this information to supply a set of derivatives at x. In this section, we will confine ourselves to the presentation of some basic concepts of automatic differentiation of first-order derivatives. For higher derivatives and a more detailed treatment, consult [12, ch.8] as well as [9].

Automatic differentiation relies on the assumption that any function evaluation is performed by evaluating sequences of "simple" unary or binary operations whose derivatives are known analytically and which in their turn are again evaluated as sequences of "simple" unary or binary operations. Binary operations include for instance addition, multiplication, division and the power operation a^b. Unary operations are for example given by the trigonometric, exponential, and logarithmic functions. Another ingredient for automatic differentiation schemes is the chain rule. For $y : \mathbb{R}^n \to \mathbb{R}^m$, $h : \mathbb{R}^m \to \mathbb{R}$,

the gradient of h in $y(x)$ is given by

$$\begin{aligned}
\nabla_x h(y(x)) &= (Dh(y(x)) \cdot Dy(x))^T \\
&= Dy(x)^T Dh(y(x))^T \\
&= \begin{bmatrix} \frac{\partial y_1}{\partial x_1}(x) & \cdots & \frac{\partial y_m}{\partial x_1}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n}(x) & \cdots & \frac{\partial y_m}{\partial x_n}(x) \end{bmatrix} \begin{bmatrix} \frac{\partial h}{\partial y_1}(y(x)) \\ \vdots \\ \frac{\partial h}{\partial y_m}(y(x)) \end{bmatrix} \\
&= \frac{\partial h}{\partial y_1}(y(x)) \begin{bmatrix} \frac{\partial y_1}{\partial x_1}(x) \\ \vdots \\ \frac{\partial y_1}{\partial x_n}(x) \end{bmatrix} + \ldots + \frac{\partial y_m}{\partial x_1}(x) \begin{bmatrix} \frac{\partial y_m}{\partial x_1}(x) \\ \vdots \\ \frac{\partial y_m}{\partial x_m}(x) \end{bmatrix} \\
&= \sum_{i=1}^{m} \frac{\partial h}{\partial y_i}(y(x)) \nabla y_i(x). \qquad (3.1)
\end{aligned}$$

Two basic modes of automatic differentiation are commonly distinguished: the *forward* and *reverse* mode. We shall illustrate their differences on a simple example.

3.1 An Example

Consider a function of 3 variables:

$$\begin{aligned}
&f : \mathbb{R}^3 \to \mathbb{R}, \\
&f(x) = (x_1 x_2 \sin x_3 + e^{x_1 x_2})/x_3. \qquad (3.2)
\end{aligned}$$

By rewriting the result $f(x)$ as a composition of intermediate unary or binary functions we extract for a given function a series of steps in a certain corresponding order for the calculation of f for any input argument. The intermediate steps are saved in interdependent subexpressions coined *intermediate variables* $x_4, x_5, \ldots$ which are distinguished from the independent input arguments x_1, x_2, x_3. For our example, these intermediate variables can be expressed as follows:

$$\begin{aligned}
x_4 &= x_1 \cdot x_2 \\
x_5 &= \sin(x_3) \\
x_6 &= \exp(x_4) \\
x_7 &= x_4 \cdot x_5 \\
x_8 &= x_6 + x_7 \\
x_9 &= x_8/x_3 = f(x).
\end{aligned}$$

The evaluation procedure can be represented in a *directed acyclic graph* (DAG), a so-called *expression graph* or a *computation graph*. For a general vector-valued function

$$r : \mathbb{R}^n \to \mathbb{R}^m$$

we designate the nodes $V = \{1, \ldots, N\}$ to represent the independent input variables ($i = 1, \ldots, n$), intermediate variables ($i = n+1, \ldots, N-m-1$) and final output variables ($i = N-m, \ldots, N$). Two nodes (i, j) form an edge from i to j if the computation of x_j requires the value of x_i. We also call x_i a *parent node* of x_j. For instance, in the evaluation

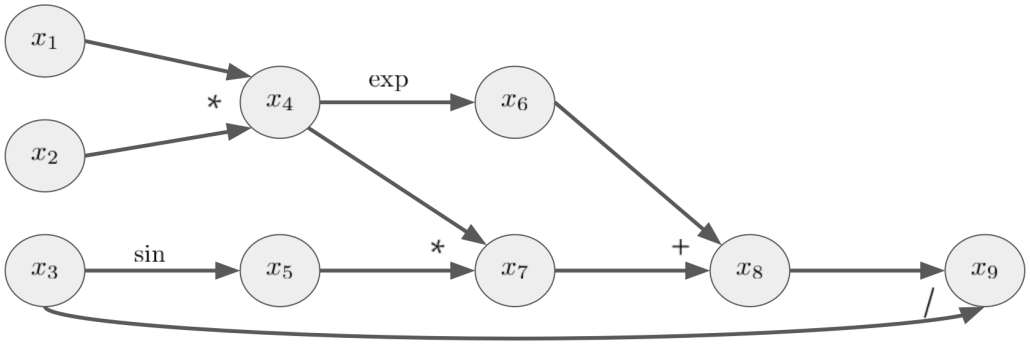


Figure 3.1: Computation graph corresponding to the evaluation of f given in (3.2).

of f ($m = 1$), the parents of x_4 are x_1 and x_2, and x_4 is a *child* of both x_1 and x_2. The corresponding computation graph for f can be seen in Figure 3.1.

The computation graph also induces a total ordering on the set of subexpressions – a so-called *topological ordering* – which we denote by $i < j$ whenever x_i is a parent of x_j.

Whenever all parent nodes of any child node are computed, the child node can in turn be computed. In this sense the computation of $f(x)$ flows through the graph from left to right. This is known as a *forward sweep*. Usually, software tools do not require the user to be able to generate such a graph or the relevant subexpressions as for this example. Instead, the construction of the intermediate variables and the computational graph is carried out by the software tool automatically.

3.2 The Forward Mode

The forward mode of automatic differentiation evaluates the directional derivative of each intermediate variable x_i by carrying its steps forward in a given direction $p \in \mathbb{R}^n$ while simultaneously evaluating x_i itself. We denote the directional derivative in p associated with each variable

$$D_p x_i := (\nabla x_i)^T p = \sum_{k=1}^{n} \frac{\partial x_i}{\partial x_k} p_k, \quad i \in \{1, \dots, m\} \tag{3.3}$$

where we settle to leave out the explicit evaluation of derivatives or partial derivatives in a point x to avoid clutter. In our example, the goal is to evaluate $\nabla f(x)^T p = D_p x_9$. Note that for $i \in \{1, 2, 3\}$, we have $D_p x_i = p_i$. The direction p is called the *seed vector*. As soon as all the values of all nodes x_i are known, it is possible to compute the corresponding derivatives $D_p x_i$ using the chain rule. For instance, if x_4, x_5 and $D_p x_4, D_p x_5$ are known we are able to calculate $D_p x_7$. Given that $x_7 = x_4 \cdot x_5$ and applying (3.1), we obtain:

$$\nabla x_7 = \frac{\partial x_7}{\partial x_4} \nabla x_4 + \frac{\partial x_7}{\partial x_5} \nabla x_5 = x_5 \nabla x_4 + x_4 \nabla x_5.$$

Taking the inner product with p and using the definition in (3.3) we arrive at

$$D_p x_7 = \frac{\partial x_7}{\partial x_4} D_p x_4 + \frac{\partial x_7}{\partial x_5} D_p x_5 = x_5 D_p x_4 + x_4 D_p x_5.$$

This calculation relies on the fact that any intermediate expression x_i is actually a function of its parent expressions $x_i = x_i((x_j)_{j \prec i})$ which are again functions of the independent input variables. In general, applying (3.1) we see that

$$\begin{aligned} D_p x_i &= \sum_{k=1}^{n} \frac{\partial x_i((x_j)_{j \prec i})}{\partial x_k} p_k \\ &= \sum_{k=1}^{n} \sum_{j \prec i} \frac{\partial x_i}{\partial x_j} \frac{\partial x_j}{\partial x_k} p_k && \text{(by (3.1))} \\ &= \sum_{j \prec i} \frac{\partial x_i}{\partial x_j} \sum_{k=1}^{n} \frac{\partial x_j}{\partial x_k} p_k \\ &= \sum_{j \prec i} \frac{\partial x_i}{\partial x_j} D_p x_j. && \text{(by (3.3))} \end{aligned}$$

Thus, in order to compute $D_p x_i$, we need only the previously computed $D_p x_j$ for all parent nodes of x_i and compute a weighted sum where the weights are given by the partial derivatives of the intermediate expression with respect to its parent expressions. In the computation graph of f, this corresponds to summing all parent values $D_p x_j$ weighing each according to a coefficient on the edge between x_j and x_i representing the value $\frac{\partial x_i}{\partial x_j}$ (see Figure 3.2). The end of the process yields $\nabla f(x)^T p = D_p x_9$.

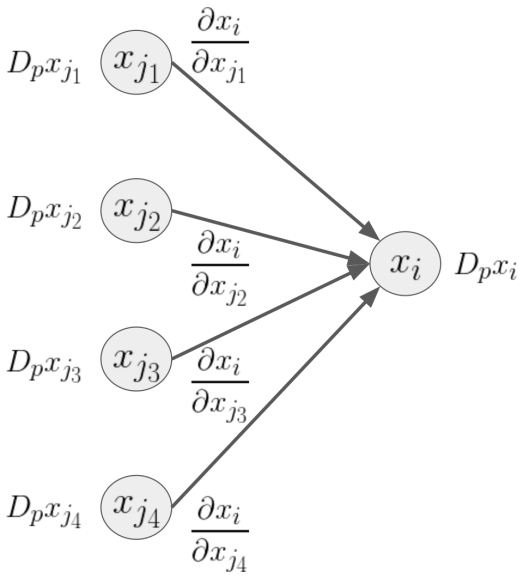


Figure 3.2: Exemplary intermediate expression node x_i with parent nodes x_{j_k}.

Furthermore, it is not necessary to store both the values of x_i and $D_p x_i$ for every node of the computation graph at once. As soon as all children of a particular node have been evaluated, there is no need to keep the values for the parent node and the storage may be freed.

The practical implementation of such automatic differentiation schemes involves the association of any scalar w appearing in the evaluation code with a scalar $D_p w$. In this way, whenever w is used in any computation, the software performs a corresponding operation of $D_p w$ based on the chain rule. For example, if w is involved in a division operation with some other value y to produce another value z

$$z \leftarrow \frac{w}{y},$$

we then use y, $D_p y$, w, and $D_p w$ to evaluate $D_p z$:

$$D_p z \leftarrow \frac{1}{y} D_p w - \frac{w}{y^2} D_p y.$$

In order to obtain the complete gradient $D_p f$ these operations need to be carried out for all $p \in \{e_1, e_2, \ldots, e_n\}$. It is clear that the cost of the evaluation of ∇f alongside f may

incur significant costs. In our example above, for instance, the evaluation of $D_p z$ gives rise to $2n$ additional multiplications and n additions. The same may go for the storage requirements seeing as for every relevant node n additional scalars $D_{e_j} x_i$, $j \in \{1, \ldots, n\}$ need to be stored for each intermediate variable x_i. Savings are possible by exploiting knowledge about which of the quantities involved may be zero such that sparse data structures can be used to store the vectors $D_{e_j} x_i$.

Concrete ways of implementing automatic differentiation would be to construct a precompiler which extends function evaluation code to automatically incorporate the evaluation of the derivatives simultaneously. Systematic operator-overloading (as can be done in C++) could also be used to extend the operations and data structures to facilitate automatic differentiation.

3.3 The Reverse Mode

The reverse mode of automatic differentiation does not evaluate the function and the corresponding gradient concurrently. Instead, all partial derivatives are recovered after the forward sweep for the computation of $f(x)$ has been executed so that the whole gradient $\nabla f(x)$ can be assembled using one so-called reverse sweep.

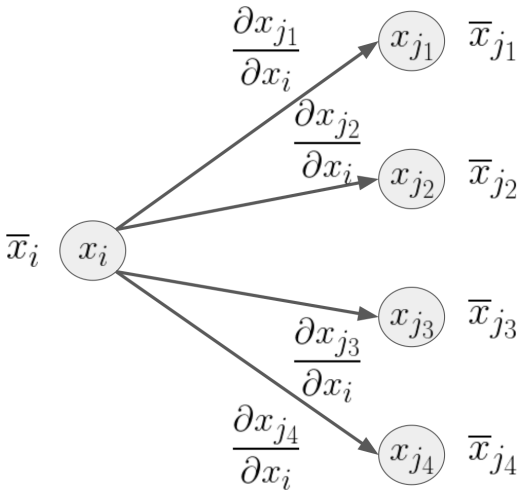


Figure 3.3: Exemplary intermediate expression node x_i with child nodes x_{j_k}.

In contrast to the forward sweep where the gradient of each *intermediate expression* x_i was computed in $D_p x_i = (\nabla x_i)^T p$, the reverse sweep computes for each intermediate expression the partial derivative $\frac{\partial f}{\partial x_i}(x)$ of f with respect to x_i. This is done by assigning a scalar $\bar{x}_i$ to each node such that $\frac{\partial f}{\partial x_i}(x)$ is the result of $\bar{x}_i$ after the reverse sweep has been executed. These $\bar{x}_i$ are also called *adjoint variables*. Let x_m be the final forward sweep evaluation node, i.e. $x_m = f(x)$. We initialize

$$\bar{x}_i = \begin{cases} 0 & , \text{ if } i \neq m \\ 1 & , \text{ else.} \end{cases}$$

Now the computation of $\frac{\partial f}{\partial x_i}$ is made to depend on the *child nodes* of x_i by recognizing that all children x_j, $j \prec i$ of x_i are themselves functions depending on x_i:

$$f(x) = f((x_j(x_i))_{j \prec i}).$$

Based on the chain rule for any node i, the partial derivative $\frac{\partial f}{\partial x_i}$ can be accumulated in a sum:

$$\frac{\partial f}{\partial x_i} = \sum_{j \prec i} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i} = \sum_{j \prec i} \frac{\partial x_j}{\partial x_i} \bar{x}_j. \tag{3.4}$$

Figure 3.3 illustrates the contrast to the forward mode.

Thus, in order to compute $\frac{\partial f}{\partial x_i} = \bar{x}_i$, $i \in \{1, \dots, n\}$, we accumulate all $\bar{x}_i$ starting from the final node $\bar{x}_m$ by successively evaluating every node's parents in the computation graph using

$$\bar{x}_i \leftarrow \bar{x}_i + \frac{\partial x_j}{\partial x_i} \bar{x}_j. \tag{3.5}$$

We gather all the contributions from the children of i and thereupon mark i as "finished". Then we update the values of each of the parents of i and continue in this fashion until all nodes are finalized. After the procedure has concluded, all components of the complete gradient can be read from $\bar{x}_i = \frac{\partial f}{\partial x_i}$ for the input nodes $i \in \{1, \dots, n\}$.

We therefore perform one forward sweep where – along with $f(x)$ – the partial derivatives $\frac{\partial x_j}{\partial x_i}$ are computed at each step where each one is associated with one edge of the computation graph. Using (3.5), the partial derivatives $\frac{\partial f}{\partial x_i}$ are then computed in the reverse sweep.

This can be illustrated by expressing our example function $f(x)$ by successively expanding the intermediate expressions x_i. Writing $x_i(x_{j_1}, \dots, x_{j_k})$ to clarify the dependence of x_i on its child node input arguments $x_{j_1}, \dots, x_{j_k}$ for some $k \in \mathbb{N}$, in the case of our example, this would result in

$$\begin{aligned} f(x_1, x_2, x_3) &= (x_1 x_2 \sin(x_3) + e^{x_1 x_2})/x_3 \\ &= (x_4(x_1, x_2) \cdot x_5(x_3) + e^{x_4(x_1, x_2)})/x_3 \\ &= (x_7(x_4, x_5) + x_6(x_4))/x_3 \\ &= x_8(x_6, x_7)/x_3 \\ &= x_9(x_3, x_8). \end{aligned}$$

Successively differentiating these expressions with respect to their relevant intermediate expressions, we obtain an expansion of the partial derivative $\frac{\partial f}{\partial x_1}$ which is then computed from left to right using the partial derivatives with respect to the intermediate expressions:

$$\begin{aligned} \bar{x}_1 = \frac{\partial f}{\partial x_1}(x) &= \underbrace{\frac{\partial f}{\partial x_4}}_{\bar{x}_4} \frac{\partial x_4}{x_1} \\ &= \left\{ \underbrace{\frac{\partial f}{\partial x_7}}_{\bar{x}_7} \frac{\partial x_7}{\partial x_4} + \underbrace{\frac{\partial f}{\partial x_6}}_{\bar{x}_6} \frac{\partial x_6}{\partial x_4} \right\} \frac{\partial x_4}{x_1} \\ &= \left\{ \left[\underbrace{\frac{\partial f}{\partial x_8}}_{\bar{x}_8} \frac{\partial x_8}{\partial x_7} \right] \frac{\partial x_7}{\partial x_4} + \left[\underbrace{\frac{\partial f}{\partial x_8}}_{\bar{x}_8} \frac{\partial x_8}{\partial x_6} \right] \frac{\partial x_6}{\partial x_4} \right\} \frac{\partial x_4}{x_1} \\ &= \left\{ \left[\left[\underbrace{\frac{\partial f}{\partial x_9}}_{\bar{x}_9} \frac{\partial x_9}{\partial x_8} \right] \frac{\partial x_8}{\partial x_7} \right] \frac{\partial x_7}{\partial x_4} + \left[\left[\underbrace{\frac{\partial f}{\partial x_9}}_{\bar{x}_9} \frac{\partial x_9}{\partial x_8} \right] \frac{\partial x_8}{\partial x_6} \right] \frac{\partial x_6}{\partial x_4} \right\} \frac{\partial x_4}{x_1}. \end{aligned}$$

A main advantage of the reverse mode is that its computational complexity is low for scalar functions $f : \mathbb{R}^n \to \mathbb{R}$ discussed in this section. Only the partial derivatives $\frac{\partial x_j}{\partial x_i}$ need to be computed in the forward sweep (just as in the forward mode), but the reverse sweep is performed only once and yields *all* components of the gradient. Given that any of the

intermediate expressions x_i depending on some previously calculated expressions x_{j_1}, x_{j_2} can be expressed as some elementary unary or binary function $\varphi_i(x_{j_1}, x_{j_2})$, the derivatives $\frac{\partial x_i}{\partial x_j} = \frac{\partial \varphi_i}{\partial x_j}(x_{j_1}, x_{j_2})$ will incur some amount ω_i of arithmetic operations themselves. Let

$$\Omega := \max_{i \in \{1, \ldots, m\}} \omega_i$$

be the maximal number of arithemtic operations performed by the evaluation of any of the partial derivatives $\frac{\partial x_i}{\partial x_j}$. We also denote by

$$c := \max_{i \in \{1, \ldots, m\}} |\{j \in \{1, \ldots, m\} \,|\, j \prec i\}|$$

the maximum number of children any node of the computation graph has. Then the forward sweep for calculating $f(x)$ and $\frac{\partial x_j}{\partial x_i}$ requires at most $2(m - n)\Omega$ arithmetic operations. The reverse sweep then adds up at most c individual values per node, so at most $2(m - 1)c$ operations. In total, this grows linearly in the number of operations needed to perform one calculation of f. A more detailed cost analysis of the forward and reverse modes of algorithmic differentiation can be found in [9, ch.4].

In contrast, the forward mode required roughly n times more arithmetic operations by doing n forward sweeps, one for each component of the gradient. This makes the reverse mode seem to perform unduly well, but as we shall see that as we go on to consider vector-valued functions $r : \mathbb{R}^n \to \mathbb{R}^m$ the costs of both procedures become more similar as m increases.

One drawback of the reverse mode is the necessity to store the entire computation graph needed for the reverse sweep. If, say, 100 million operations are performed and each node could be saved using only 20 bytes, then the graph would already grow to 2 gigabytes in size. Some methods, such as *checkpointing* [9, ch.12] reduce the size requirements to save the entire graph by performing partial forward and reverse sweeps such that only portions of the graph are needed instead of the entire structure.

3.4 Calculating Jacobians of Vector Functions

The forward mode for vector-valued functions can be performed quite similarly to the scalar case. We simply extend the scalar case by adding as many final nodes as are needed, each representing the value of $r_j(x)$ for $j = 1, \ldots, m$. We assign to each of these nodes a variable $D_p r_j = (\nabla r_j)^T p$ which is the jth component of $J(x)p$. Assembling all these quantities yields $J(x)p$. As in the scalar case, we can assemble the entire Jacobian by setting $p \in \{e_1, \ldots, e_n\}$ and calculating the n quantities $D_{e_j} x_i$ for every node i simultaneously. As in the section 2.1.2, we can again apply the colouring techniques to sparse Jacobians in order to bring down on the number of vectors p that need to be evaluated.

The reverse mode on the other hand can be executed by reducing the vector-valued problem to a scalar-valued problem. To that end, we choose some seed vector $q \in \mathbb{R}^m$ and

apply the reverse mode to $r(x)^T q$. For any vector $v \in \mathbb{R}^m$, we have

$$\begin{aligned} D(r(x)^T q)[v] &= D(\langle \cdot, q \rangle)(r(x)) \circ D(r)(x)[v] \\ &= (\langle \cdot, q \rangle) D(r)(x)[v] \\ &= \langle J(x)v, q \rangle \\ &= \langle v, J(x)^T q \rangle \\ &= \langle J(x)^T q, v \rangle. \end{aligned}$$

Hence, $\nabla(r(x)^T q) = J(x)^T q$. So the reverse mode applied to $r(x)^T q$ yields a Jacobian-transpose-vector product which can be implemented by setting the initial seed variables $\bar{x}_j$ corresponding to the final values r_j, $j \in \{1, \ldots, m\}$ to q_j:

$$\bar{x}_j = q_j$$

since $\frac{\partial r^T q}{\partial r_j}(x) = q_j$. Thus, at the conclusion of the reverse sweep, the $\bar{x}_i$ corresponding to the independent input variables x_i will contain the components of $J(x)^T q$.

Again, we can make use of the colouring techniques of section 2.1.2 for sparse Jacobians and we can combine both approaches to extract individual rows and columns using the reverse and the forward modes respectively.

3.5 Application: Gradient Descent Method for a Finite-Dimensional Minimization Problem

For an exemplary application which combines both the discretization approaches of section 2 in order to approximate the solution to an infinite-dimensional minimization problem, let us consider a version of the *minimal surface problem* on a unit square $\Omega =]0,1[^2$:

$$\begin{cases} \text{Find } z \in \mathcal{C}^1(\Omega) \cap \mathcal{C}(\overline{\Omega}) \text{ such that} \\ \mathcal{S}(z) := \displaystyle\int_\Omega \sqrt{1 + \left(\frac{\partial z}{\partial x}(x,y)\right)^2 + \left(\frac{\partial z}{\partial y}(x,y)\right)^2}\, d\lambda^2(x,y) \to \min \\ z = g \quad \text{on } \Gamma \subseteq \Omega \end{cases} \tag{3.6}$$

for a given function $g : \Gamma \to \mathbb{R}$ defined on a subset $\Gamma \subseteq \Omega$. Here, $\mathcal{S}(z)$ gives the value of the surface area of the submanifold[5] defined by the graph of z on Ω. We shall not concern ourselves with the closer analysis of (3.6), specifically whether solutions to the problem exist or are unique. Instead, our interest is to obtain a finite-dimensional minimization problem from (3.6) on which to test a gradient-based optimization algorithm and observe illustrative solutions of the discretized problem.

We discretize $\mathcal{S}(z)$ using a composite Newton-Cotes quadrature formula (see [7, ch.10.2, ch.10.5.2]). We set

$$g(x,y) := \sqrt{1 + \left(\frac{\partial z}{\partial x}(x,y)\right)^2 + \left(\frac{\partial z}{\partial y}(x,y)\right)^2}.$$

[5] This is proven as a special case ($n = 2$) in Lemma A.2 in the Appendix.

Dividing both the axes of Ω into n subintervals $[s_{k-1}, s_k]$ and $[t_{k-1}, t_k]$ of equal lengths $\frac{1}{n}$ for $k \in \{1, \dots, n\}$ and applying a Newton-Cotes quadrature rule of order m for any function along the axes, we obtain as an approximation for $\mathcal{S}(z)$:

$$\begin{aligned}\mathcal{S}(z) &= \sum_{k=1}^{n}\sum_{l=1}^{n}\int_{s_{k-1}}^{s_k}\int_{t_{k-1}}^{t_k} g(x,y)\,dxdy \\ &\approx \sum_{k=1}^{n}\sum_{l=1}^{n}\frac{1}{n^2}\sum_{i=0}^{m}\sum_{j=0}^{m} c_i c_j g(x_i, y_j)\end{aligned}$$

with weights $c_i, c_j \in \mathbb{R}$ corresponding to the applied Newton-Cotes quadrature rule. Formally, we consider grid functions

$$z \in \overline{G}_h = \{z_{i,j} \,|\, z(x_i, y_j) := z_{i,j},\ i,j \in \{1, \dots, d\}\} \cong \mathbb{R}^N$$

where $d := m+(n-1)(m-1)$ is the number of individual grid nodes in either direction, $N = d^2$ and $h = \frac{1}{d-1}$ is the distance between two individual grid nodes in either direction. Grid functions define approximations to graphs of the sought-after functions $z \in \mathcal{C}^1(\Omega) \cap \mathcal{C}(\overline{\Omega})$ by approximating z on the grid nodes (x_i, y_j). We write $z \in \overline{G}_h$ or equivalenty $z \in \mathbb{R}^N$ if we order the nodes (x_i, y_j) lexicographically (from bottom to top, left to right alternating on the unit square). Approximating $g(x_i, y_j)$ using the forward difference formula (2.10), we obtain

$$g(x_i, y_j) \approx f(x_i, y_j) = \sqrt{1 + \left(\frac{z(x_{i+1}, y_j) - z(x_i, y_j)}{h}\right)^2 + \left(\frac{z(x_i, y_{j+1}) - z(x_i, y_j)}{h}\right)^2}.$$

Our finite-dimensional target function $\hat{Z} : \mathbb{R}^N \to \mathbb{R}$ is then defined by

$$\hat{Z}(z) = \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n}\sum_{i=0}^{m}\sum_{j=0}^{m} c_i c_j f(x_i, y_j). \tag{3.7}$$

Thus, our discretized problem is

$$\begin{cases} \text{Find } z \in \overline{G}_h \text{ such that} \\ \hat{Z}(z) := \dfrac{1}{n^2}\displaystyle\sum_{k=1}^{n}\sum_{l=1}^{n}\sum_{i=0}^{m}\sum_{j=0}^{m} c_i c_j f(x_i, y_j) \to \min \\ z = g \quad \text{on } \Gamma_h \subseteq \overline{\Omega}_h \end{cases} \tag{3.8}$$

One method to search for a minimum of $\hat{Z}$ is simply a gradient-descent method with an Armijo line search procedure (see [1] for further details). We skip investigating sufficient conditions for the convergence of this procedure. In fact, one important assumption – the compactness of the sublevel set $N(\hat{Z}, \hat{Z}(z_0))$ of the initial value $z_0 \in \overline{G}_h$ – is violated since $\hat{Z}(z_0) = \hat{Z}(z_0 + c)$ for any constant $c \in \mathbb{R}$ causing the sublevel set to be unbounded for any $z_0 \in \overline{G}_h$.

A spirit of bold optimism took hold of me at this point and I tried my luck anyway. The discretization (3.7) was implemented[6] in `python` using the symbolic framework of

[6]Exemplary scripts can be downloaded from https://github.com/Anemometer/minimal_surface.

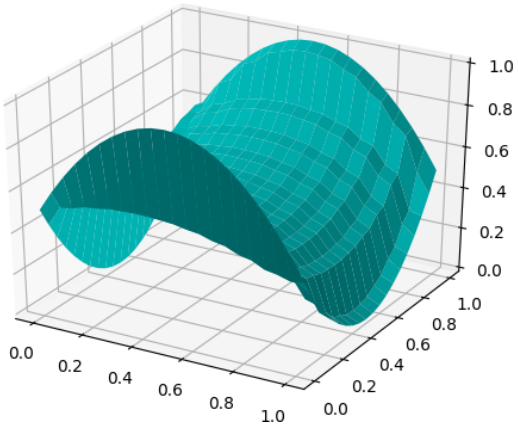
the nonlinear optimization and optimal control library `CasADi` [2]. The restriction $z = g$ on Γ for $z \in \overline{G}_h$ was realized by writing $z = (z_k)_{k=1}^N \in \mathbb{R}^N$ such that

$$z(x_i, y_j) = g(x_i, y_j) \iff e_{\ell(i,j)}^T z = g(x_i, y_j)$$

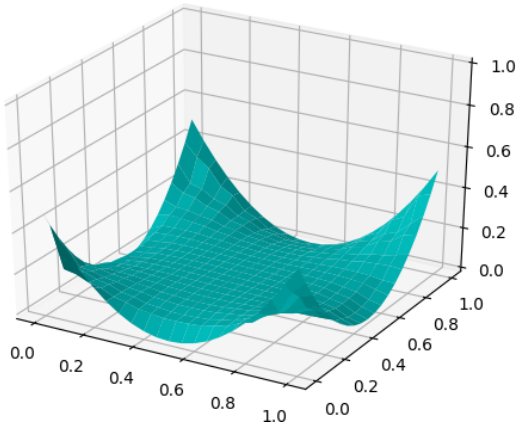
where $\ell(i, j)$ gives the lexicographical index of the index pair $(i, j) \in \{1, \ldots, d\}^2$. The restriction $z = g$ in the minimization problem (3.8) can thus be transformed into an unconstrained minimization problem by means of a kernel space transformation (see [1, ch.6.3]). Various configurations of Γ and g were tested. Four of the end results for different combinations of quadratic boundary functions

$$\Gamma = \partial\Omega$$
$$g(x) = \pm a \cdot (x - 0.5)^2 + c$$

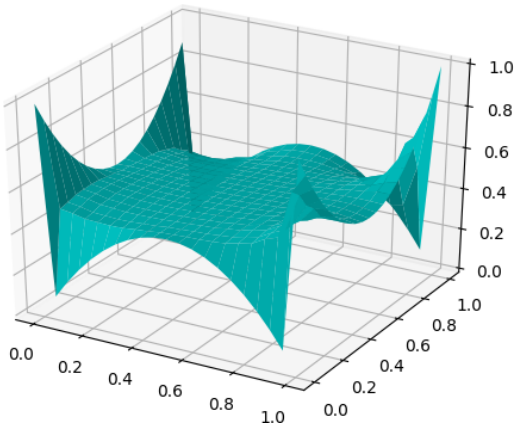
for various constants a, c as well as one version including a line segment in the interior of Ω with the same constraint g can be found in Figure 3.4.



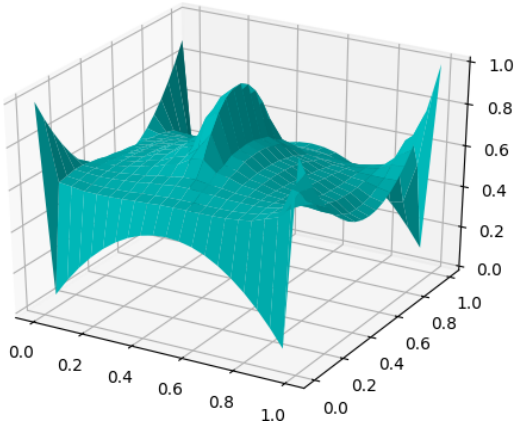
(a) Iterate for $\Gamma = \partial\Omega$ after 4770 iterations.



(b) Iterate for $\Gamma = \partial\Omega$ after 1790 iterations.



(c) Iterate for $\Gamma = \partial\Omega$ after 1560 iterations.



(d) Iterate for $\Gamma = \partial\Omega \cup \{(x, 0.5) \,|\, x \in [0.25, 0.75]\}$ after 1460 iterations.

Figure 3.4: Surface plots of the final iterates of the gradient descent procedure for the discretized minimal surface problem (3.8) for various constraint configurations after the convergence criterion $\|\nabla \hat{Z}(z_k)\|_2 < 10^{-4}$ was satisfied.

A Appendix

Lemma A.1. *Let $A \in \mathbb{K}^{m\times n}$ with $\mathbb{K}$ being either $\mathbb{R}$ or $\mathbb{C}$ and $\|\cdot\|_1 : \mathbb{K}^n \to \mathbb{R}, x \mapsto \|x\|_1 := \sum_{i=1}^n |x_i|$ the 1-norm on $\mathbb{K}^n$. Then for the induced operator norm holds*

$$\|A\|_1 := \max_{\|x\|_1=1} \|Ax\|_1 = \max_{1\le j\le n} \sum_{i=1}^m |a_{ij}| = \max_{1\le j\le n} \|Ae_j\|_1. \tag{A.1}$$

Proof.

We prove (A.1) by establishing both $\|A\|_1 \le \max_{1\le j\le n} \sum_{i=1}^m |a_{i,j}|$ and $\|A\|_1 \ge \max_{1\le j\le n} \sum_{i=1}^m |a_{i,j}|$. First, observe that

$$\begin{aligned}
\|A\|_1 &= \max_{\|x\|_1=1} \|Ax\|_1 = \max_{\|x\|_1=1} \sum_{i=1}^m \left| \sum_{j=1}^n a_{ij} x_j \right| \\
&\le \max_{\|x\|_1=1} \sum_{i=1}^m \sum_{j=1}^n |a_{ij}||x_j| = \max_{\|x\|_1=1} \sum_{j=1}^n \left(|x_j| \sum_{i=1}^m |a_{ij}| \right) \\
&\le \max_{\|x\|_1=1} \sum_{j=1}^n \left(|x_j| \max_{1\le k\le n} \sum_{i=1}^n |a_{ik}| \right) \\
&= \max_{1\le j\le n} \sum_{i=1}^m |a_{ij}| \max_{\|x\|_1=1} \|x\|_1 \\
&= \max_{1\le j\le n} \sum_{i=1}^m |a_{ij}|.
\end{aligned}$$

On the other hand, let k be the index of the column with the largest sum:

$$k = \arg\max_{1\le j\le n} \sum_{i=1}^m |a_{ij}|,$$

i.e. $\max_{1\le j\le n} \sum_{i=1}^m |a_{ij}| = \sum_{i=1}^m |a_{ik}| = \|Ae_k\|_1$. But it also holds $\|A\|_1 \ge \|Ae_k\|_1$. Thus follows (A.1). □

Lemma A.2. *Let $\Omega \subseteq \mathbb{R}^n$ be open and $\gamma : \Omega \to \mathbb{R}$ continuously differentiable such that a submanifold $\mathcal{M}$ is locally represented as the graph of γ such that there exists a finite number of local immersions of the form $\varphi_i : U_i \to \mathbb{R}^{n+1}$*

$$\varphi_i(x) = \begin{bmatrix} x \\ \gamma(x) \end{bmatrix}$$

for $U_i \subseteq \Omega$ open, $i \in I \subseteq \mathbb{N}$ finite. Then the surface area spanned by $\mathcal{M}$ in $\mathbb{R}^{n+1}$ is given by

$$\begin{aligned}
\mathrm{vol}_n(\mathcal{M}) &= \int_{\mathcal{M}} 1 \mathrm{dS}(x) \\
&= \sum_{i\in I} \int_{\mathcal{M}} \alpha_i(x) \mathrm{dS}(x) \\
&= \sum_{i\in I} \int_{\Omega} \alpha_i(x) \sqrt{1 + \sum_{j=1}^n \left(\frac{\partial \gamma}{\partial x_i}(x) \right)^2} \mathrm{d}\lambda^n(x)
\end{aligned} \tag{A.2}$$

for a suitable partition of unity $(\alpha_i)_{i\in I}$ on $\mathcal{M}$.

Proof.

Since $\mathrm{vol}_n(\mathcal{M})$ splits into a sum over each individual immersion φ_i, it is sufficient to show (A.2) for one single $U \subseteq \Omega$ open. In that case, we have

$$\mathrm{vol}_n(\varphi(U)) = \int_{\varphi(U)} 1 \, dS(x) = \int_U \sqrt{\det(D\gamma(x)^T D\gamma(x))} \, d\lambda^n(x) \tag{A.3}$$

with the Gram matrix $D\gamma(x)^T D\gamma(x)$. We see that

$$D\gamma(x) = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & 1 \\ \frac{\partial\gamma}{\partial x_1}(x) & \cdots & & \frac{\partial\gamma}{\partial x_n}(x) \end{bmatrix}$$

implying

$$\begin{aligned} D\gamma(x)^T D\gamma(x) &= \begin{bmatrix} 1 + \left(\frac{\partial\gamma}{\partial x_1}(x)\right)^2 & \frac{\partial\gamma}{\partial x_1}(x)\frac{\partial\gamma}{\partial x_2}(x) & \frac{\partial\gamma}{\partial x_1}(x)\frac{\partial\gamma}{\partial x_3}(x) & \cdots & \frac{\partial\gamma}{\partial x_1}(x)\frac{\partial\gamma}{\partial x_n}(x) \\ \frac{\partial\gamma}{\partial x_1}(x)\frac{\partial\gamma}{\partial x_2}(x) & 1 + \left(\frac{\partial\gamma}{\partial x_2}(x)\right)^2 & \frac{\partial\gamma}{\partial x_2}(x)\frac{\partial\gamma}{\partial x_3}(x) & \cdots & \frac{\partial\gamma}{\partial x_2}(x)\frac{\partial\gamma}{\partial x_n}(x) \\ \frac{\partial\gamma}{\partial x_1}(x)\frac{\partial\gamma}{\partial x_3}(x) & & & & \\ \vdots & & & & \vdots \\ \frac{\partial\gamma}{\partial x_1}(x)\frac{\partial\gamma}{\partial x_n}(x) & & & & 1 + \left(\frac{\partial\gamma}{\partial x_n}(x)\right)^2 \end{bmatrix} \\ &= \begin{bmatrix} e_1 + \frac{\partial\gamma}{\partial x_1}(x)\nabla\gamma(x) & \cdots & e_n + \frac{\partial\gamma}{\partial x_n}(x)\nabla\gamma(x) \end{bmatrix} \end{aligned}$$

Taking the determinant, we have

$$\det(D\gamma(x)^T D\gamma(x)) = \det\begin{bmatrix} e_1 + \frac{\partial\gamma}{\partial x_1}(x)\nabla\gamma(x) & \cdots & e_n + \frac{\partial\gamma}{\partial x_n}(x)\nabla\gamma(x) \end{bmatrix}. \tag{A.4}$$

Since $\det$ is multilinear and alternating, expanding all sums in (A.4) we retain only the summands $\det\begin{bmatrix} e_1 & \cdots & e_n \end{bmatrix} = 1$ and

$$\begin{aligned} &\det\begin{bmatrix} e_1 & \cdots & e_{i-1} & \frac{\partial\gamma}{\partial x_i}(x)\nabla\gamma(x) & e_{i+1} & \cdots & e_n \end{bmatrix} \\ &= \frac{\partial\gamma}{\partial x_i}(x)\det\begin{bmatrix} e_1 & \cdots & e_{i-1} & \nabla\gamma(x) & e_{i+1} & \cdots & e_n \end{bmatrix} \\ &= \frac{\partial\gamma}{\partial x_i}(x)\sum_{j=1}^{n}\det\begin{bmatrix} e_1 & \cdots & e_{i-1} & \frac{\partial\gamma}{\partial x_j}(x)e_j & e_{i+1} & \cdots & e_n \end{bmatrix} \\ &= \frac{\partial\gamma}{\partial x_i}(x)\frac{\partial\gamma}{\partial x_i}(x) = \left(\frac{\partial\gamma}{\partial x_i}(x)\right)^2 \end{aligned}$$

for $i \in \{1, \ldots, n\}$ since all other summands are zero by the alternating property of $\det$. Thus, (A.4) simplifies to

$$\det(D\gamma(x)^T D\gamma(x)) = 1 + \sum_{i=1}^{n}\left(\frac{\partial\gamma}{\partial x_i}(x)\right)^2.$$

Plugging this expression into (A.3) we obtain

$$\mathrm{vol}_n(\varphi(U)) = \int_U \sqrt{1 + \sum_{i=1}^{n} \left(\frac{\partial \gamma}{\partial x_i}(x)\right)^2}\, d\lambda^n(x)$$

as required.

□

References

[1] W. Alt. *Nichtlineare Optimierung: Eine Einführung in Theorie, Verfahren und Anwendungen*. 1st ed. Vieweg, 2002.

[2] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* (In Press, 2018).

[3] P.G. Ciarlet. *Mathematical Elasticity: Three-Dimensional Elasticity*. Vol. 1. Amsterdam: Elsevier, 1988.

[4] P.G. Ciarlet. *Linear and Nonlinear Functional Analysis with Applications*. Society for Industrial and Applied Mathematics, 2013.

[5] Thomas F. Coleman and Jorge J. Moré. "Estimation of sparse jacobian matrices and graph coloring problems". In: *SIAM Journal of Numerical Analysis* 20.1 (1983), pp. 187–209.

[6] Thomas F. Coleman and Jorge J. Moré. "Estimation of sparse hessian matrices and graph coloring problems". In: *Mathematical Programming* 28.3 (1984), pp. 243–270.

[7] W. Dahmen and A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. 1st ed. Springer-Verlag Berlin Heidelberg, 2006.

[8] O. Forster. *Analysis 1*. 9th ed. Vieweg+Teubner Verlag, 2006.

[9] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics, 2008.

[10] J.E. Dennis Jr. and R.B. Schnabel. "A View of Unconstrained Optimization". In: *Optimization*. Ed. by G.L. Nemhauser and A.H.G. Rinnooy Kan and M.J. Todd. Vol. 1. Amsterdam: Elsevier, 1989. Chap. 1, pp. 1–72.

[11] Herbert B. Keller. *Numerical methods for two-point boundary-value problems*. Dover Publications, 1992.

[12] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2nd ed. Springer, 2006.