Differentiation

Continuity

at a point iff $\lim_{h\to 0} f(x+h) = f(x)$

Given f, g, cont. functions, the following functions are cont

$f+g$	$\lvert f\rvert$
cf for $c \in \mathbb{R}$	$exp(f)$
f^n for $n \in \mathbb{N}$	f^α for $\alpha \in \mathbb{N}, f$ strictly positive
fg	$f/g, g \neq 0$
$f \circ g$	$\log f, f$ strictly positive
$max(f,g)$	

Differentiation in 1D

at a point x $\iff$ $\lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$ .uses the same rules as in table above, but not max or $|f|$
Rules for derivative combinations are kinda obv. Extension of quotient rule:

$$\frac{d}{dx}\left(\frac{f}{g^k}\right) = \frac{g\frac{df}{dx} - kf\frac{dg}{dx}}{g^{k+1}}$$

Taylor's Theorem

For $k >= 0$, and $f : D \to \mathbb{R}$ is 'smooth'

$$\begin{aligned} f(x) = f(x_0) &+ (x-x_0)\frac{\mathrm{d}f}{\mathrm{d}x}(x_0) + \cdots \\ &+ \frac{(x-x_0)^k}{k!}\frac{\mathrm{d}^k f}{\mathrm{d}x^k}(x_0) \\ &+ \frac{(x-x_0)^{k+1}}{k+1!}\frac{\mathrm{d}^{k+1} f}{\mathrm{d}x^{k+1}}(\xi) \Leftarrow \text{Lagrange remainder} \\ &\text{for some } \xi \in (x_0, x) \end{aligned}$$

Partial differentiation

Consider $f : D \to \mathbb{R}$, for $D \subseteq \mathbb{R}^n$ a partial derivative is, e.g. $\frac{\partial f}{\partial x}$ or $\frac{\partial^2 f}{\partial x \partial y}$ (Clairaut's theorem gives us that $\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x}$ if both are cont). Define the Hessian of f as:

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

n-D differentiation

For a function $f : D \to \mathbb{R}$ for $D \subseteq \mathbb{R}^n$:

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n}\right)^T \leftarrow \text{ a } \textbf{column} \text{ vector}$$

Linearity, the product rule and the quotient rule all work as expected from the 1D case.

The chain rule does as well, at least for $g : \mathbb{R} \to \mathbb{R}$ with f as above. But if $f : \mathbb{R}^m \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}$, then

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}(g \circ \boldsymbol{f}) = \boldsymbol{J}(\boldsymbol{f})^T \left(\frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}} \circ \boldsymbol{f} \right)$$

Useful derivatives include:

Function		Derivative
$\boldsymbol{a}^T\boldsymbol{x},\ \boldsymbol{x}^T\boldsymbol{a}$	$\to$	$\boldsymbol{a}$
$\boldsymbol{x}^T\boldsymbol{x}$	$\to$	$2\boldsymbol{x}$
$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$	$\to$	$(\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{x} = 2\boldsymbol{A}\boldsymbol{x}$ (if A symmetric)

Jacobian

Given $f : \mathbb{R}^m \to \mathbb{R}^n$, we can consider f as a **column** vector of scalar-valued functions. Then the Jacobian is its 'derivative'

$$\boldsymbol{J}(\boldsymbol{f}) = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

So $J_{i,j} = \dfrac{\partial f_i}{\partial x_j}$

Note, obviously, that $\boldsymbol{H}(f) = \boldsymbol{J}(\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}})^T$. Combinations of Jacobian matrices:

$\boldsymbol{J}(\boldsymbol{f}+\boldsymbol{g})$	$=$	$\boldsymbol{J}(\boldsymbol{f}) + \boldsymbol{J}(\boldsymbol{g})$	
$\boldsymbol{J}(c\boldsymbol{f})$	$=$	$c\boldsymbol{J}(\boldsymbol{f})$	(for const c)
$\boldsymbol{J}(\boldsymbol{A}\boldsymbol{f})$	$=$	$\boldsymbol{A}\boldsymbol{J}(\boldsymbol{f})$	(for const matrix $\boldsymbol{A}$)
$\boldsymbol{J}(\boldsymbol{f}^T\boldsymbol{g})$	$=$	$\boldsymbol{g}^T\boldsymbol{J}(\boldsymbol{f}) + \boldsymbol{f}^T\boldsymbol{J}(\boldsymbol{g})$	(**dot product rule**)
$\boldsymbol{J}(f\boldsymbol{g})$	$=$	$\boldsymbol{g}\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}^T + f\boldsymbol{J}(\boldsymbol{g})$	(prod. of scalar, vector)
$\boldsymbol{J}(\boldsymbol{g} \circ \boldsymbol{f})$	$=$	$(\boldsymbol{J}(\boldsymbol{g}) \circ \boldsymbol{f})\boldsymbol{J}(f)$	(partial chain rule)
$\boldsymbol{J}(\boldsymbol{x})$	$=$	$\boldsymbol{I}$	
$\boldsymbol{J}(\boldsymbol{A}\boldsymbol{x})$	$=$	$\boldsymbol{A}$	

Taylor's Theorem in nD

Given $f : D \to \mathbb{R}$ for $D \subseteq \mathbb{R}^n$:

$$f(\boldsymbol{x}) = f(\boldsymbol{x_0}) + (\boldsymbol{x} - \boldsymbol{x_0})^T \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x^*}) \ \textbf{(0th order)}$$

$$f(\boldsymbol{x}) = f(\boldsymbol{x_0}) + (\boldsymbol{x} - \boldsymbol{x_0})^T \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0}) \ \textbf{(1st order)} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x_0})^T \boldsymbol{H}(f)(\boldsymbol{x^*})(\boldsymbol{x} - \boldsymbol{x_0})$$

$$f(\boldsymbol{x}) = f(\boldsymbol{x_0}) + (\boldsymbol{x} - \boldsymbol{x_0})^T \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_0}) \ \textbf{(2nd order)} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x_0})^T \boldsymbol{H}(f)(\boldsymbol{x^*})(\boldsymbol{x} - \boldsymbol{x_0}) + e_3$$

Optimisation

To find: $\arg\min_{x \in F} f(\mathbf{x})$ for an **objective function** f, on a **feasible set** F.
If $F = \mathbb{R}^n$ then optimisation is **unconstrained**. Otherwise, it is **constrained**, and F is normally defined as

$$F = \{\mathbf{x} \in \mathbb{R}^n | g_i(\mathbf{x}) = 0, ..., h_i(\mathbf{x}) \geq 0, ...\}$$

If $F = \emptyset$ then the problem is **infeasible/inconsistent**, and if there is no minimum then it is **unbounded**.

1D

i.e. for $f : \mathbb{R} \to \mathbb{R}$. First we need $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$. We then need to find global minima:

$$\text{global minima} \subseteq \text{local minima} \subseteq \text{stationary points}$$

For constrained optimisation you also need to **check endpoint values.**
If k is the index of the first non-zero derivative, then

$$x_0 \text{ is local } \begin{cases} \begin{cases} \text{minimum} & \text{if } \frac{\mathrm{d}^k f}{\mathrm{d}x^k} > 0 \\ \text{maximum} & \text{if } \frac{\mathrm{d}^k f}{\mathrm{d}x^k} < 0 \end{cases} & \text{if } k \text{ is even} \\ \text{stationary point} & \text{if } k \text{ is odd} \end{cases}$$

Of course, if $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_0) > 0$ then x_0 is immediately a local min.

nD, unconstrained

i.e. for $f : \mathbb{R}^n \to \mathbb{R}$. First we need $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = 0$.

$$\boldsymbol{H}(f)((x_0) \text{ is positive definite} \Rightarrow \boldsymbol{x_0} \text{ is a minimum}$$

For a symmetric matrix $\boldsymbol{A}$

$$\boldsymbol{A} \text{ is } \left\{ \begin{array}{c} \text{positive definite} \\ \text{positive } \textbf{semi-}\text{definite} \\ \text{negative definite} \\ \text{negative } \textbf{semi-}\text{definite} \end{array} \right\} \text{ if } x^T \boldsymbol{A} x \left\{ \begin{array}{c} > \\ \geq \\ < \\ \leq \end{array} \right\} 0$$

If $\boldsymbol{A}$ is 2-by-2 then you can replace $x^T \boldsymbol{A} x$ by **eigenvalues** or **pivots** above.
Also: $\boldsymbol{A}$ is positive (semi-)definite $\Leftrightarrow -\boldsymbol{A}$ is negative (semi-)definite.

For a positive (semi-)definite matrix $\boldsymbol{A}$, so is :

$\boldsymbol{A} + \boldsymbol{B}$	for $\boldsymbol{B}$ positive semi-definite
$c\boldsymbol{A}$	for $c \in \mathbb{R}$
$\boldsymbol{A}^{-1}$	which exists **if** $\boldsymbol{A}$ is pos def
$\boldsymbol{C}^T \boldsymbol{A} \boldsymbol{C}$	for full rank $\boldsymbol{C}$ for pos def, otherwise any $\boldsymbol{C}$

Convexity

A **set** $D \subseteq \mathbb{R}^n$ is convex $\iff \forall \boldsymbol{x_1}, \boldsymbol{x_2} \in D, \forall \alpha \in (0,1) \quad (1-\alpha)\boldsymbol{x_1} + \alpha \boldsymbol{x_2} \in D$.
A **function** $f : D \to \mathbb{R}^n$ on convex $D \subseteq \mathbb{R}^n$ is convex $\iff \forall \boldsymbol{x_1}, \boldsymbol{x_2} \in D, \alpha \in (0,1)$

$$f((1-\alpha)\boldsymbol{x_1} + \alpha \boldsymbol{x_2}) \leq (1-\alpha) f(\boldsymbol{x_1}) + \alpha f(\boldsymbol{x_2})$$

(strict convexity defs have strict inequalities)
If we have a convex f, then

$$\text{global minima} = \text{local minima} = \text{stationary points}$$

But of course in constrained optimisation we still need to **check the boundary**:
for a **function** $f : D \to \mathbb{R}^n$ on convex $D \subseteq \mathbb{R}^n$ with cont. second-order derivatives

$$f \text{ (-/strictly) convex } \Leftrightarrow \boldsymbol{H}(f) \text{ positive (semi/-)definite}$$

Convex	Concave (strict is *)
$y = ax + b$	$y = ax + b$
* $exp(x)$	* $logx$
	$\lvert x\rvert^p$ for * $p > 1$ (if $p = 1$ only concave)

For f, g (strictly) convex functions on convex $D \subseteq \mathbb{R}^n$, so are

$$\begin{aligned} f + g & \\ cf & \quad \forall c \in \mathbb{R} \\ x \mapsto f(\boldsymbol{A}x + b) & \quad \text{(for strict convexity } \boldsymbol{A} \text{ full rank)} \\ max(f, g) & \\ expf & \end{aligned}$$

Tricks
If g is a strictly increasing function,

$$\arg\min_{x \in F} f(x) = \arg\min_{x \in F} g(f(x))$$

e.g. if f is a product (and a positive function) then $\log f$ is a sum, and sometimes easier to diff.
If g is a 1-1 function,

$$\arg\min_{x \in F} f(x) = g\left(\arg\min_{x \in F} f(g(x))\right)$$

e.g. to minimise $\cos(\exp(x))$ we need points where $\exp(x) = n\pi$.

Equality constraints
Use the **Lagrangian**, as it has the same stationary points on the constraint as f:

$$\Lambda(\boldsymbol{x}, \lambda_1, \lambda_2 ...) = f(\boldsymbol{x}) - \lambda_1 g_1(\boldsymbol{x}) - \lambda_2 g_2(\boldsymbol{x}) ...$$

Then differentiate and eliminate λ to get solutions, then classify them as follows.

- Check if f is unbounded (i.e. if there is even a global min)
- evaluate the stationary points to find the min.

Inequality constraints
The question is if each constraint $h_i(\boldsymbol{x})$ is **slack** (> 0) or **tight** ($= 0$) Guess (with reasoning) which are so, and solve all options for remaining tight/slack constraints, and check that the ones assumed to be slack are actually slack.

Numerical integration

in 1D

Note $m = \frac{a+b}{2}$,
$\underline{D_k} = \min_{x \in (a,b)} \frac{\mathrm{d}^k f}{\mathrm{d}x^k}$
$\overline{D_K} = \max_{x \in (a,b)} \frac{\mathrm{d}^k f}{\mathrm{d}x^k}$

Midpoint Rule:

$$\begin{aligned} M_1[f,a,b] &= (b-a)f(m) \\ err(M_1)[f,a,b] &\leq -\frac{1}{24}\boldsymbol{(b-a)^3}\underline{D_2} \\ M_n[f,a,b] &= \sum_n^{i=1} M_1[f,x_{i-1},x_i] \\ &= \frac{b-a}{n}\sum_n^{i=1} f\left(\frac{x_{i-1}+x_i}{2}\right) \\ err(M_n)[f,a,b] &= \sum_n^{i=1} err(M_1)[f,x_{i-1},x_i] \leq -\frac{(b-a)^3}{24n^2}\underline{D_2} \end{aligned}$$

Trapezium Rule:

$$\begin{aligned} T_1[f,a,b] &= \frac{b-a}{2}(f(a)+f(b)) \\ err(T_1)[f,a,b] &\leq \frac{\boldsymbol{(b-a)^3}}{12}\overline{D_2} \\ T_n[f,a,b] &= \sum_n^{i=1} T_1[f,x_{i-1},x_i] \\ &= \frac{b-a}{2n}\left(f(x_0)+2f(x_1)+\cdots+2f(x_{n-1})+f(x_n)\right)) \\ err(T_n)[f,a,b] &\leq -\frac{(b-a)^3}{12n^2}\overline{D_2} \end{aligned}$$

Simpson's Rule:

Note n must be even.

$$\begin{aligned} S_2[f,a,b] &= \frac{b-a}{6}(f(a)+4f(m)+f(b)) \\ err(S_2)[f,a,b] &\leq \frac{\boldsymbol{(b-a)^5}}{2880}\overline{D_4} \\ S_n[f,a,b] &= \sum_{n/2}^{i=1} S_2[f,x_{2i-2},x_{2i}] \\ &= \frac{b-a}{3n}\left(f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+\cdots+4f(x_{n-1})+f(x_n)\right)) \\ err(S_n)[f,a,b] &\leq -\frac{(b-a)^5}{180n^4}\overline{D_4} \end{aligned}$$

Note all these composite methods do about the same amount of work (all eval. f roughly n times)

in nD

The 1D methods above can be extended (i.e. evaluating f on grids in n dimensions) but that means their error converges only at $O(n^{-2/d})$ and $O(n^{-4/d})$, which is very slow. So we use **Monte Carlo Integration**:
Given N random vectors $\boldsymbol{X}_1, ...\boldsymbol{X}_N$

$$MC_N[f,R] = \text{Area}(R)\frac{1}{N}\sum_i f(\boldsymbol{X_i})$$

Useful properties include:

- $\mathrm{E}[MC_N[f,R]] = \int_R f(\boldsymbol{x})d\boldsymbol{x}$ (i.e. unbiased)
- $\mathrm{Var}[MC_N[f,R]] = V/N$ for a constant
- $\mathrm{P}[|err(MC_N)[f,R]| \geq \epsilon] \to 0$ as $N \to \infty$, for any $\epsilon > 0$ (i.e. consistent
- $err(MC_N)[f,R]$ approaches the Normal distribution $N(0, \frac{V}{N})$ for large N.

So the error (the standard error $\sqrt{\frac{V}{N}}$) decreases as $O(N^{-1/2})$ regardless of dimension, which is slower than, e.g. Simpson's but much better if d is large.
To use this, we need to generate random vectors in R, so the simplest way to do that is choose an S such that $R \subseteq S$ and random vectors in S can be easily generated, and then discard vectors in $S \backslash R$. If necessary, we can calculate Area(R) as Area(S) $\times$ the proportion of accepted samples from S.
Techniques to improve this: **importance sampling**, **control variates** ($\int_R f(\boldsymbol{x})d\boldsymbol{x} = \int_R (f(\boldsymbol{x} - \hat{f}(\boldsymbol{x}))d\boldsymbol{x} + \int_R \hat{f}(\boldsymbol{x})d\boldsymbol{x})$, **stratified sampling**

Accuracy

	$\tilde{u} - u$	is the **error**
For $u \in \mathbb{R}$:	$\lvert\tilde{u} - u\rvert$	is the **absolute error**
	$\lvert\tilde{u} - u\rvert / \lvert u\rvert$	is the **relative error**

If $u \in \mathbb{R}^n$ replace $|x|$ with the Euclidean norm $\|x\|$.

Error

Rounding error only roughly 15 sig digits can be stored

Machine epsilon, ϵ the smallest quantity st. every $x \in \mathbb{R}$ can be stored as $\tilde{x}$, a floating point with relative error $\leq \epsilon$

For double precision $\epsilon \approx 2^{-53} \approx 1.1 \times 10^{-16}$

Most methods run with accuracy to within a few multiples of ϵ. Others will run to $\sqrt{\epsilon}$ which is bigger than ϵ.

Catastrophic cancellation occurs if $a \approx b$, then small relative errors in a or b can cause large relative errors in $a - b$. Avoid this by rewriting formulae to avoid subtraction of this kind

Truncation error error when approximating an infinite procedure with a finite one (e.g. a Taylor poly vs the orig. function)

Roundoff error is introduced by approximating a real number with a stored (floating point) number

Propagation is when a small error is magnified by the following procedures (often called **instability**), e.g. solving $\boldsymbol{Ax} = \boldsymbol{b}$

Rates of convergence

Given a algorithm that seeks $\boldsymbol{x^*}$ with iterative guesses $\boldsymbol{x_1}, \boldsymbol{x_2}, ...$, the absolute error at the n-th step is $\epsilon_n = x_n - x^*$.

If $\lvert\epsilon_n\rvert \to 0$, convergence is			
	linear	$\frac{\lvert\epsilon_{n+1}\rvert}{\lvert\epsilon_n\rvert} \to \alpha \in (0,1)$	c^n for $c < 1$
	sublinear	$\frac{\lvert\epsilon_{n+1}\rvert}{\lvert\epsilon_n\rvert} \to 1$	(throughout)
	logarithmic (is sublinear)	$\frac{\lvert\epsilon_{n+2}-\epsilon_{n+1}\rvert}{\lvert\epsilon_{n+1}-\epsilon_n\rvert} \to 1$	$1/n$
	superlinear	$\frac{\lvert\epsilon_{n+1}\rvert}{\lvert\epsilon_n\rvert} \to 0$	c^{n^2}
	order-q ($q > 1$)	$\frac{\lvert\epsilon_{n+1}\rvert}{\lvert\epsilon_n^q\rvert} \to \alpha > 0$	c^{2^n} is order-2

Order-2 convergence is often called quadratic. It is very useful when comparing algorithms to know that:

> If A_n converges with order q, then A_{2n} converges with order q^2.

Rootfinding

Termination

Forward error bounds:

$\lvert x_n - x_{n-1}\rvert < tol$	a small change in abs. error
$\lvert x_n - x_{n-1}\rvert < tol\lvert x_n\rvert$	a small change in relative error
$\lvert x_n - x_{n-1}\rvert < tol(1 + \lvert x_n\rvert)$	measures both abs. and rel

OR Backward error bound: $|f(x_n)| < tol$ function close to zero (backward error bound)

Practical bounds: (must include)

$n = N$	iteration ran out of time
a step was ill-defined	so an error must be raised

Interval bisection

Given $m_n = \frac{a_n+b_n}{2}$,
If $f(a_n)f(m_n) < 0$ set (a_{n+1}, b_{n+1}) to be (a_n, m_n), otherwise (m_n, b_n).
It is slow, but guarantees convergence given a bracket (use either maths or test increasing brackets of the form $(-2^k, 2^k)$), but this method does not generalise to higher dimensions.
Convergence is linear, as $\epsilon_{n+1} = \frac{1}{2}\epsilon_n$, and we need $n = \left\lceil \log \frac{b_0-a_0}{2tol} \right\rceil$

Newton's method in 1D

Uses 1st-order Taylor poly as an approximation, and finds the root of that.

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)}$$

Problems

- $\frac{\mathrm{d}f}{\mathrm{d}x}(x_n) = 0$ means the iteration is undefined
- it can diverge (if $\frac{\mathrm{d}f}{\mathrm{d}x}$ is close/eq. to 0)
- it can converge to a different root
- it can also get stuck in a loop

Error

By the same Taylor expansion,

$$x^* - x_n \approx \frac{f(x_n)}{\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)} = x_{n+1} - x_n$$

So we can make one more iteration to get an **a posteriori error estimate**.

Given

$$A_I(c) = \frac{\max_{\beta \in I} \left| \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\beta) \right|}{\min_{\alpha \in I} \left| \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(\alpha) \right|}$$

- For an interval $I = (x^* - c, x^*)$, if $x_0 \in I$ and $\frac{cA_I(c)}{2} < 1$, Newton's method converges at least quadratically to x^*.
- Given a bracket for x^* of the form $(m - c, m + c)$, Newton's method converges at least quadratically starting at m, if $\frac{cA_J(c)}{2} < 1$ for the interval $J = (m - 2c, m + 2c)$.
- Such an interval J exists if $\frac{\mathrm{d}f}{\mathrm{d}x}(x^*) \neq 0$.

If f has a very small derivative at the root (so f is **ill-conditioned**) then there is possibility for significant roundoff error.

Secant method, 1D

Uses **linear interpolation** and solves that approximation.

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

Only needs 1 eval. of f per step, as we can remember $f(x_{n-1})$.
Can fail if $f(x_n) \approx f(x_{n-1})$, and a useful termination condition is $|f(x_n) - f(x_{n-1})| < tol|f(x_n)|$ to avoid this.

Error

Same conditions as Newton's Method, to get order-ϕ convergence, for ϕ between 1.5 and 1.7. Remember that two steps of the secant method have order $\phi^2 > 2$ convergence, which is faster than Newton's method (so if Newton is slower than two secant steps, use this instead).

Newton's method in dD

We are solving $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ for $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^d$. To get this method split $\boldsymbol{f}$ into d functions $\mathbb{R}^d \to \mathbb{R}$ and apply the 1st order Taylor poly, to get:

$$x_{n+1} = x_n - (\boldsymbol{J}(\boldsymbol{f})(\boldsymbol{x_n}))^{-1} \boldsymbol{f}(\boldsymbol{x_n})$$

or, which is about 3 times **faster**, solve for $\boldsymbol{\Delta x}$ in $(\boldsymbol{J}(\boldsymbol{f})(\boldsymbol{x_n}))\boldsymbol{\Delta x} = -\boldsymbol{f}(\boldsymbol{x_n})$ and set $\boldsymbol{x_{n+1}} = \boldsymbol{x_n} + \boldsymbol{\Delta x}$
If $\boldsymbol{J}(\boldsymbol{f})(x_n)$ is singular, there may not be a solution, so we must stop. The cost per iteration is $O(d^3)$ to solve for Δx and $O(d^2)$ to evaluate the Jacobian.
We can make this less fragile if we insist that $\|\boldsymbol{f}(\boldsymbol{x_{n+1}})\| < \|\boldsymbol{f}(\boldsymbol{x_n})\|$ (†), and if it is not true, set $\boldsymbol{x_{n+1}} = \boldsymbol{x_n} + \lambda \boldsymbol{\Delta x}$ for $\lambda \in (0, 1)$. Damping with small λ will always satisfy (†), but may only be linear convergence.

Quasi-Newton methods

Essentially we want to approximate $\boldsymbol{J_n}$, with $\hat{\boldsymbol{J}}_{\boldsymbol{n}}$. This must satisfy the secant equation: ($\boldsymbol{y_r} = \boldsymbol{f}(\boldsymbol{x_r})$)

$$y_n = y_{n-1} + \hat{\boldsymbol{J}}_{\boldsymbol{n}}(\boldsymbol{x_n} - \boldsymbol{x_{n-1}})$$

And also must not change for vectors orthogonal to $x_n - x_{n-1}$ (Trid).
A starting approximate Jacobian is needed (either $-\boldsymbol{I}$ or $\boldsymbol{J}(\boldsymbol{f})(\boldsymbol{x_0})$), and we can optimise further by tracking the inverse of the approximation and only updating that.
Termination is similar to Newton's, and it may need more steps, but they will likely be faster.

Numerical optimisation

Similar termination conditions to rootfinding:

- $|x_n - x_{n-1}| < tol(1 + |x_n|)$ - loc. of approx min didn't change much (might signify step length too short, but good for Newton-like methods)
- $|f(x_n) - f(x_{n-1})| < tol(1 + |f(x_n)|$ - val. of approx min didn't change much (dangerous if fun. flat far from min)
- $|\frac{\mathrm{d}f}{\mathrm{d}x}(x_n)| < tol$ - when gradient close to zero (only applicable if methods computes gradient)
- $n = N$ or ill-defined steps.

Warning: The relative error in **loc. of min** is $O(\sqrt{\epsilon}) > O(\epsilon)$ (from Taylor's rule if relative error in value is less than ϵ).

Golden section search, 1D:

A **bracket** is a triple (a, b, c) with $a < b < c$,and $f(a) > f(b) < f(c)$, which must contain at least 1 local minimum.
Refinement process:

- choose $z \in (a, c)$ and find the appropriate triple out of a, b, c, z that is a bracket.
- Choose z using the golden ratio: if (a, b) is the larger of (a, b) and (b, c), $b - z = \phi(b - a)$. Also $z - a = c - b$.
- If the previous bracket followed the same ratio, then we have ϕ as the golden ratio.

This is linear convergence, with one eval. of f per iteration - the bracket shrinks by ≈ 0.618 per iter. A starting bracket is required - e.g using $(0, (\phi + 1)^n, (\phi + 1)^{n+1})$

Gradient descent in d dimensions

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} + \alpha_n \boldsymbol{d_n}$$

Criteria for $\boldsymbol{d_n}$ and α_n: (Note $\boldsymbol{g_n} = \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x_n})$)

1. $\boldsymbol{d_n}$ is a **descent direction**, so $\boldsymbol{g_n}^T \boldsymbol{d_n} < 0$
2. $\boldsymbol{d_n}, \boldsymbol{d_{n-1}}, ...$ do not veer wildly
3. α_n seeks to roughly minimise $f(\boldsymbol{x_n} + \alpha_n \boldsymbol{d_n})$
4. the method does not stop - i.e. $\sum_1^\infty \alpha_n = \infty$

Options for d_n:

- **coordinate gradient descent**: cycle through unit vectors $\pm e_i$. Fast, needs no derivatives, but naive
- **gradient descent**: $\boldsymbol{d_n} = -\boldsymbol{g_n}$. Satisfies conditions (indeed best sol.)
- **conjugate grad descent**: directions that satisfy $\boldsymbol{d_n}^T \boldsymbol{H}(f)(\boldsymbol{x_n}) \boldsymbol{d_m} = 0$ for $m < n$
- in machine learning when obj. fun is an average loss, use only part of the data to find d_n (deterministic/random).

Options for α_n:

- Constant: simple, but often slow progress or not actually descent
- **backtracking**: choose an initial step length a'_n and multiply by $0 < \rho < 1$ until it is suitable.
- Good starting step length: $\alpha'_n = \alpha_{n-1} \frac{\boldsymbol{g_{n-1}}^T \boldsymbol{d_{n-1}}}{\boldsymbol{g_n}^T \boldsymbol{d_n}}$
- **Armijo rule** for suitable α_n: $f(\boldsymbol{x_n} + \alpha_n \boldsymbol{d_n}) < f(\boldsymbol{x_n}) + \sigma \alpha_n \boldsymbol{g_n}^T \boldsymbol{d_n}$ for (roughly) $\sigma \in (10^{-4}, 10^{-1})$.

Since we have the gradient $\boldsymbol{g_n}$ it is sensible to use it in the termination conditions (e.g. as $\|\boldsymbol{g_n}\| < tol(1 + \|\boldsymbol{g_0}\|)$)

Newton methods in d dimensions

Use a 2nd order Taylor expansion and minimise that:

$$x_{n+1} = x_n - (\boldsymbol{H}(f)(\boldsymbol{x_n}))^{-1} \boldsymbol{g_n}$$

Much more work, as Hessian calculation is $O(d^2)$ and $O(d^3)$ to solve for Δx for the inverse, which is undefined if $\boldsymbol{H}(f)(\boldsymbol{x_n})$ is singular.
Also the Newton direction isn't necessarily a descent direction. (it might find a maximum).
If f has a well-behaved local minimum $\boldsymbol{x^*}$ $\boldsymbol{H}(f)(\boldsymbol{x_n})$ should be pos def near $\boldsymbol{x^*}$ so Newton will be quadratic conv. (not always true, e.g. if Hess only semi-def).
Can try to fix the Hessian by adding $\lambda \boldsymbol{I}$ for some λ.

Quasi-Newton methods

Approximate the the Hessian somehow.