H4: nonlinear equations

4.1 introduction	
nonlinear phenomena	= effects that aren't directly proportional to their causes > nonlinear equations can be represented by: $\mathbf{f}(\mathbf{x}) = \mathbf{y}$. > subtract y to find: $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. > root finding problem
4.2 number of solutions	
solutions of a nonlinear problem	= correspond to the points where a curved hyperplane f(x) intersect > no general statements about number of solutions
multiple roots	a smooth function f has multiple roots if: $f(x^*) = f'(x^*) = f''(x^*) = \cdots = f^{(m-1)}(x^*) = 0$ then x* is a root of multiplicity m
4.3 sensitivity	
sensitivity of a root	= if x* is a root, how much does x* for small changes to the parameters of f
condition number	= parameter for sensitivity in one dimension = $\frac{1}{\|f'(x^*)\|}$ > if f'(x) is small near x*, the error of the root is big At a multiple root x*, f'(x*) = 0 > condition number is infinite
condition number in multiple dim.	In multiple dim. this is the Jacobian **J** > $\|\mathbf{J}_f^{-1}(x)\|$.
4.4 convergence rates and stopping criteria	
convergence rate	= the effectiveness with which a certain algorithm reaches its solution def: Let $\mathbf{e}_k = \mathbf{x}_k - x^*$ be the error at iteration k, where $\mathbf{x}_k$ is the approximate solution at iteration k and x^* the (usually unknown) true solution. An iterative method is said to converge with rate r if $\lim_{k\to\infty} \frac{\|\mathbf{e}_{k+1}\|}{\|\mathbf{e}_k\|^r} = C$ for some finite constant $C > 0$.
cost of solving a system	= depends on number of iterations + amount of iterations needed
types of convergence	• $r = 1$ and $C < 1$: *linear* convergence • $r > 1$: *superlinear* convergence • $r = 2$: *quadratic* convergence • $r = 3$: *cubic* convergence
stopping criterion	look at the relative change in the solutions: $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| / \|\mathbf{x}_k\| < \varepsilon,$ with ε the *error tolerance*

4.5 solving nonlinear equations in one dimension	
for equations in one dimension, we seek a x* for a function f: $\mathbb{R}\to\mathbb{R}$ such that f(x*)=0	
4.5.1 bisection method	
bisection method	There might not exist a machine number x* for which f(x*) is exactly 0 > search for a bracket [a,b] where the sign changes Begin with an initial bracket > iteratively reduce its length until the desired accuracy is reached nl: for each iteration, evaluate the function at the midpoint of the interval > discard half the interval, based of the sign of this value
> convergence	bisection method makes no use of the magnitude of the function values > is certain to converge, but very slowly >> each iteration, the bound on possible error is reduced by half > convergence is linear with r=1 and C=0.5 error: Given a starting interval $[a, b]$, the length of the interval after k iterations is $(b-a)/2^k$, so that achieving an error tolerance of ε requires $$\left\lceil n = \log_2\left(\frac{b-a}{\varepsilon}\right)\right\rceil$$ $$\left\lceil \iff \varepsilon = \left(\frac{b-a}{2^n}\right)\right\rceil$$ iterations, regardless of the particular function f involved.
4.5.2 fixed-point iteration	
fixed point problem	For a function g: $\mathbb{R}\to\mathbb{R}$ > a fixed point is a point x for which g(x) = x ie: finding an intersection between g and the diagonal line y=x
fixed-point iteration	For solving nonlinear equations we can use iterations of the form $$x_{k+1} = g(x_k)$$ where g is a functions so that its fixed points are solutions for f(x)=0 >> there are multiple methods that use fixed-point iteration
> convergence	look at the derivative of g in a solution x* > if x*=g(x) and ‖g'(x)‖<1, then the iterative scheme is locally convergent else, ‖g'(x)‖>1, the scheme diverges for every initial value different from x* The convergence rate of the iterative scheme is linear with C=‖g'(x)‖ nl: the smaller C, the faster the convergence >> ideally ‖g'(x)‖=0, in which case the Taylor expansion gives us: $$g(x_k) - g(x^*) = g''(\xi_k)(x_k - x^*)^2/2$$ with ξ_k between x_k and x^*. This yields $$\lim_{k\to\infty} \frac{\|e_{k+1}\|}{\|e_k\|^2} = \frac{g''(x^*)}{2}$$ In this case the *rate of convergence becomes quadratic* In the next sections we'll see methods to systematically choose g to reach this quadratic convergence.

4.5.3 Newton's method	
newton's method	For the truncated Taylor series, a linear function of h that approximates f near a given x: $$f(x+h) \approx f(x) + hf'(x)$$ for f'(x)!=0, its zero is determined by: $$h = -f(x)/f'(x),$$ >> repeat this method in an iterative scheme > systematic way of transforming a linear equation f(x)=0 in a fixed point problem x=g(x) where: $$g(x) = x - f(x)/f'(x)$$ thus: 1. **Start with an initial guess:** Choose an initial guess x_0 for the root of the function $f(x) = 0$. 2. **Compute the function value and its derivative:** Evaluate the function $f(x)$ and its derivative $f'(x)$ at the current guess x_n. 3. **Update the guess:** Use the formula $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ to calculate a new estimate x_{n+1} for the root. 4. **Repeat:** Repeat steps 2 and 3 until the difference between consecutive estimates $\lvert x_{n+1} - x_n \rvert$ is sufficiently small or until a specified number of iterations is reached.
> convergence	look at the derivative of g(x): $$g'(x) = f(x)f''(x)/(f'(x))^2$$ • For simple roots ($f(x^*) = 0$ and $f'(x^*) \neq 0$), $g'(x^*) = 0$. Thus the asymptotic convergence rate of Newton's method is quadratic. • For a multiple root with multiplicity m, it is only linearly convergent, with constant $C = 1 - (1/m)$.
4.5.4 secant method	
secant method	= Newton's method, but replace its derivative by: $$f'(x_k) = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$ > approximate f by the secant line through the previous two estimates > and take the zero of this function as the best approximate solution >> easier, because you don't have to explicitly determine the derivative of f
> convergence	drawback: - you need two guesses for the iteration to start - converging more slowly nl: subquadratically, but faster than linear with r=1.618 (compared to Newtons method)
4.5.5 inverse interpolation	
inverse interpolation	instead of fitting polynomial to values $f(x_k)$ as function of the values x_k do the opposite > fir a polynomial p to the values x_k as a function of the values $f(x_k)$ > the approximate solution than is p(0) >> most used: inverse quadratic interpolation ie: fit a parabola through the values obtained at the last 3 iterations > like to secant method, only requires one additional function evaluation per iteration > requires little more memory to fit a parabola convergence rate of r=1.839

4.5.6 root finding in SciPy	
root finding in SciPy	for 1D function: Brent method via optimize.brentq > is a safeguard method that combines safety of bracket method with high convergence of inverse quadratic interpolation
4.5.7 roots of polynomial functions	
multiple roots	polynomials have multiple roots > for p(x) of degree n, we want to find all n roots > several methods: • Use one of the methods shown above to find one root x_1 and then deflate the polynomial $p(x)$ to $p(x)/(x-x_1)$ which has a degree that is one lower and repeat the process. Note that it's a good idea to zoom in on each of the obtained roots using the approximate values used this way to avoid any numerical errors introduced in the deflating process. • Use a dedicated (complex) routine specifically designed for this purpose. These work by isolating the roots of a polynomial in the complex plane, and then refining in a way similar to the bisection method to zoom in on each of the roots. Their complexity is beyond the scope of this course. • Form the **companion matrix** of the given polynomial and use an eigenvalue routine to find its eigenvalues, which are also the roots of the polynomial.
4.6 systems of nonlinear equations	
system of nonlinear equations	more difficult because • A much wider range of behavior is possible, so we don't get as far with theoretical analysis of the existence and number of solutions. • There is no simple way to bracket a desired solution. • Computational overhead increases rapidly with the dimension of the problem. Most methods in 1D don't generalize for multiple dimensions > however, Newton's method does: For a differentiable vector function $\mathbf{f}$, the truncated Taylor series reads: $$\mathbf{f}(\mathbf{x}+\mathbf{s}) \approx \mathbf{f}(\mathbf{x}) + \mathbf{J}_f(\mathbf{x})\mathbf{s}$$, where $\mathbf{J_f(x)}$ is the Jacobian matrix of $\mathbf{f}$ with elements $$\{\mathbf{J_f(x)}\}_{ij} = \frac{\partial f_i(\mathbf{x})}{\partial x_j}$$ If $\mathbf{s}$ satisfies the linear system $\mathbf{J}_f(\mathbf{x})\mathbf{s} = -\mathbf{f}(\mathbf{x})$, then $\mathbf{x}+\mathbf{s}$ is taken as an approximate zero of $\mathbf{f}$. Essentially, Newton's method replaces a system of nonlinear equations with a system of linear equations, but as the solutions of both systems are not identical, the process must be repeated until the desired accuracy is reached. If the Jacobian of the function is not available, there exist more advanced methods which estimate the Jacobian based on function evaluations, similar to how the secant method works in 1 dimension. The computational cost of Newton's method in n dimensions is substantial: • Evaluating the Jacobian matrix (or approximating it) requires n^2 function evaluations. • Solving the system $\mathbf{J}_f(\mathbf{x})\mathbf{s} = -\mathbf{f}(\mathbf{x})$, for instance using LU-factorization, costs $\mathcal{O}(n^3)$ operations.
in SciPy	use optimize.root