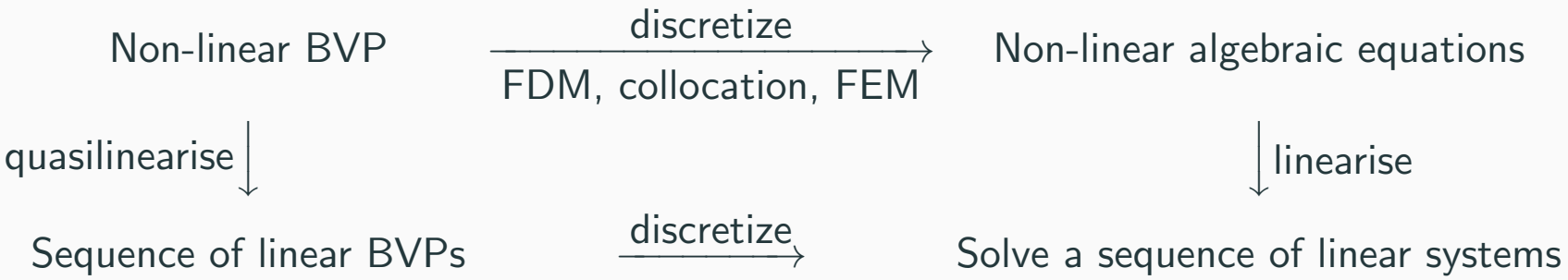
MAST90026 Computational Differential Equations: Week 4

Jesse Collis
Modified from Hailong Guo (2022)

Semester 1 2024

The University of Melbourne

Handling nonlinearities



Demonstration

Consider a simple non-linear BVP:

$$u'' + 1 - u^2 = 0, \qquad u(0) = 0, u(1) = 1,$$

with $N + 1$ mesh points or $N - 1$ internal points.

FDM:

$$\frac{u_{j-1} - 2u_j + u_{j+1}}{h^2} + 1 - u_j^2 = 0, \qquad j = 2, \ldots N, \qquad u_1 = 0,\ u_{N+1} = 1.$$

Matrix form of FDM

System of nonlinear equations:

$$\frac{1}{h^2}\begin{pmatrix} -2 & 1 & 0 & \cdots & \cdots \\ 1 & -2 & 1 & 0 & \vdots \\ 0 & 1 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \cdots & \cdots & \cdots & -2 \end{pmatrix} \begin{pmatrix} u_2 \\ \vdots \\ \vdots \\ \vdots \\ u_N \end{pmatrix} + \begin{pmatrix} 1-u_2^2 \\ \vdots \\ \vdots \\ \vdots \\ 1-u_N^2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ -\frac{1}{h^2} \end{pmatrix}$$

This is a nonlinear system of algebraic equations

$$\vec{F}(\vec{u}) = A\vec{u} + 1 - \vec{u} \odot \vec{u} - \vec{b} = \vec{0}.$$

Method for solving nonlinear equations

Fixed point or Picard iteration (easy to implement but usually linearly convergent)

Newton's Method (need f' but quadratically convergent)

Bisection (globally convergent but slow)

Secant method (don't need f' but super linear convergence)

Fixed point (Picard) iteration

Idea of Picard iteration: rewrite $f(x) = 0$ as $x = g(x)$.

Condition on convergence: g is contractive i.e. $|g(x) - g(y)| < C|x - y|$ for some $C < 1$.

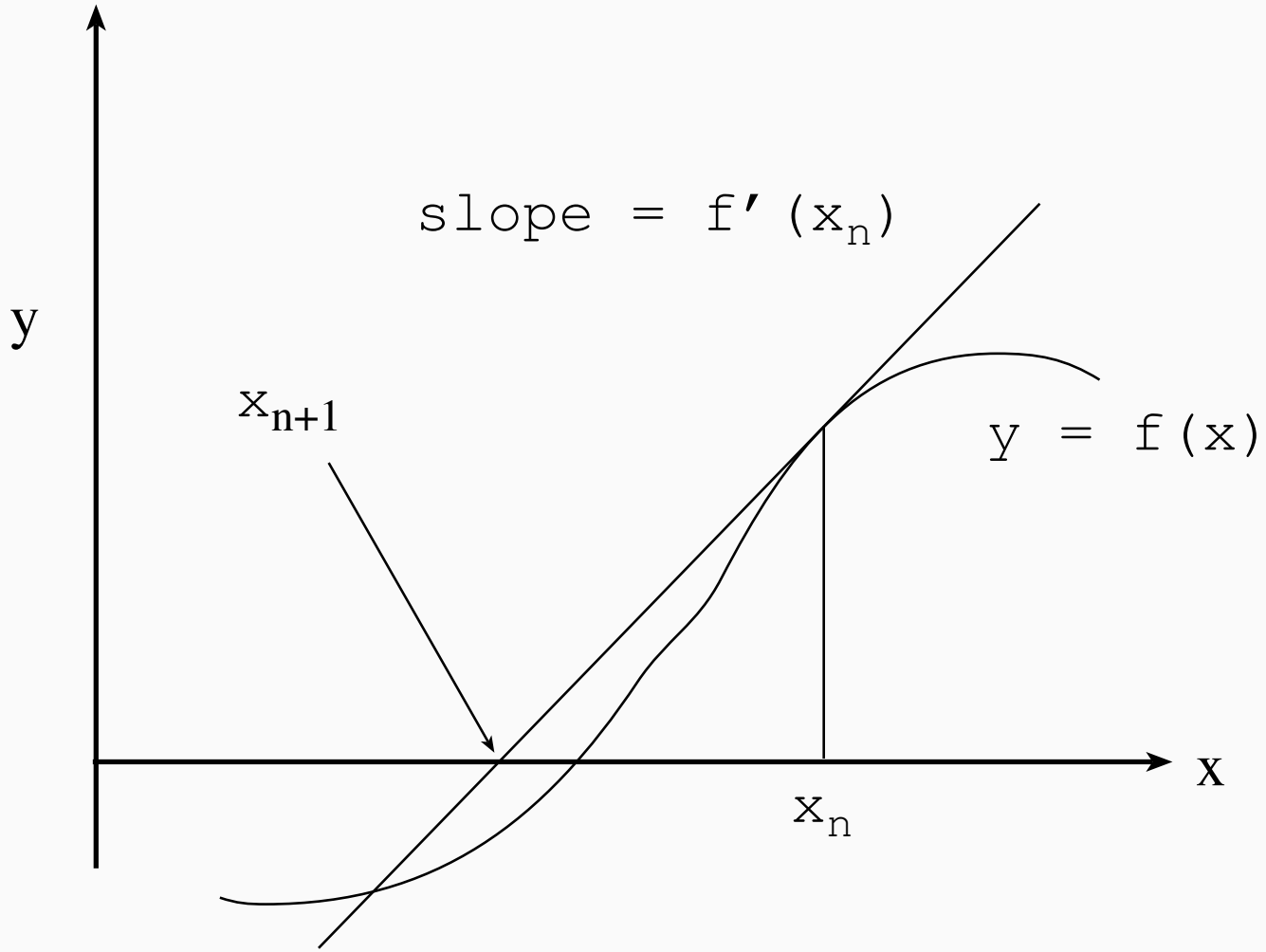
Sufficient condition: $|g'| < 1$.

Higher-dimensional: rewrite $\vec{F}(\vec{u}) = \vec{0}$ as $\vec{u} = \vec{G}$

Our example:

$$\vec{u}_{N+1} = \vec{A}^{-1}[\vec{b} - (1 - \vec{u}_N \odot \vec{u}_N)].$$

Idea of Newton's method



Derivation of Newton's method using Taylor series

Using Taylor series:

$$f(x_{n+1}) \approx f(x_n) - f'(x_n)(x_{n+1} - x_n),$$

or written as an update

$$x_{n+1} = x_n + \Delta x_n, \quad \text{where} \quad \Delta x_n = \frac{-f(x_n)}{f'(x_n)}.$$

Quadratic convergence, i.e. $e_{n+1} \leq K e_n^2$ but not always convergent.

2D Newton's method

Consider $F(u, v) = 0$, and $G(u, v) = 0$.

Taylor series expansion:

$$0 = F(u_{n+1}, v_{n+1}) \approx F(u_n, v_n) + \left.\frac{\partial F}{\partial u}\right|_n (u_{n+1} - u_n) + \left.\frac{\partial F}{\partial v}\right|_n (v_{n+1} - v_n),$$

$$0 = G(u_{n+1}, v_{n+1}) \approx G(u_n, v_n) + \left.\frac{\partial G}{\partial u}\right|_n \Delta u_n + \left.\frac{\partial G}{\partial v}\right|_n \Delta v_n,$$

Matrix form

$$\begin{pmatrix} \left.\frac{\partial F}{\partial u}\right|_n & \left.\frac{\partial F}{\partial v}\right|_n \\ \left.\frac{\partial G}{\partial u}\right|_n & \left.\frac{\partial G}{\partial v}\right|_n \end{pmatrix} \begin{pmatrix} \Delta u_n \\ \Delta v_n \end{pmatrix} = - \begin{pmatrix} F(u_n, v_n) \\ G(u_n, v_n) \end{pmatrix} = -\vec{F}(\vec{u}_n).$$

General Newton's method

Derive using Taylor series expansion:

$$0 = \mathbf{F}(\mathbf{u}_{n+1}) \approx \mathbf{F}(\mathbf{u}_n) + \mathbf{F}'(\mathbf{u}_n)(\mathbf{u}_{n+1} - \mathbf{u}_n).$$

Newton update

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \delta_n$$

where δ_n solves the linear system

$$J(\mathbf{u}_n)\,\delta_n = -\mathbf{F}(\mathbf{u}_n)$$

Here $J(\theta) \equiv F'(\theta) \in \mathbb{R}^{m\times m}$ is the Jacobian matrix with elements

$$J_{ij}(\mathbf{u}) = \frac{\partial}{\partial u_j} F_i(\mathbf{u}),$$

where $F_i(\mathbf{u})$ is the i th component of the vector-valued function $\mathbf{F}$.

Discussion of Newton's method

Pros: If it convergent, it is quadratic convergent.

Cons:

- Need to compute the $N \times N$ matrix J of partial derivative.
- Need to solve a sequence of $N \times N$ linear systems ($O(N^3)$ work).

Back to our Example

Considering $u'' + 1 - u^2 = 0$ again

$$F_i = \sum_j A_{ij} u_j + (1 - u_i^2) - b_i = 0,$$

$$J_{ij} = \frac{\partial F_i}{\partial u_j} = A_{ij} - 2u_j \delta_{ij}.$$

In this case, J was sparse (tridiagonal) so it's not too hard to compute or solve with. It's a good idea to declare it sparse or use spdiags to create it.

Applicability of Newton's method

Need $J_{ij} = \frac{\partial F_i}{\partial u_j}$ which is often expensive because it has N^2 elements. It is also often hard to evaluate the Jacobian analytically.

- Finite difference approximation for the partial derivative.

$$\frac{\partial F_i}{\partial u_j} \approx \frac{F_i(u_j + \delta) - F_i(u_j)}{\delta},$$

  where $\delta \sim C\sqrt{r}$ where C is a constant and r is the unit roundoff. This retains quadratic convergence down to $e_{n+1} \sim r$.
- Use automatic differentiation (Ref: Andreas Griewank) which finds the action of the Jacobian on F.

Efficiency of Newton's method

Idea: evaluate J, which has N^2 elements as seldom as possible whilst also solving $J\Delta u = -f$ ($O(N^3)$ operations) as seldom as possible.

Motivation: solve $J\Delta u = -f$ in two steps: 1. factorise $J = LU$ into upper and lower triangular matrices; 2. solve the triangular systems which takes $O(N^2)$ time. S modify Newton's method to evaluate J only every so often instead of every step.

- Chord method: start at $\vec{u_0}$ and only evaluate and factorise $J(\vec{u_0})$ once and then solve

$$(LU)_0 \Delta u_n = -F(\vec{u_n}), \quad u_{n+1} = u_n + \Delta u_n,$$

  repeatedly. much cheaper but only has linear convergence.
- Re-evaluate J and refactorise every m step. ($m = 1$ is Newton's method and $m = \infty$ is the chord method).
- inexact Newton methods and truncated Newton methods: solve the systems approximately using an iterative solver.

Convergence of Newton's method

Newton's method only solves equations if $\vec{u}_0$ is close enough to the solution $\vec{u^*}$. Question: how do we find $\vec{u}_0$?

Damped Newton methods which seek to globalize Newton's method by increasing the basin of attraction, r.

To do this, we observe that often the step size is too big causing us to over shoot the root, even though we have the direction right. To do this we replace the Newton step (or modified Newton step) by:

$$\vec{u}_{n+1} = \vec{u}_n + \lambda \Delta u_n,$$

where $\lambda \in (0, 1)$ is the damping factor.

We can modify this further by varying λ from step to step, always choosing λ_n such that $\|F\|$ falls by a "sufficient" amount (Ref: Armijo 1966).

Quasi-linearisation

Consider the boundary value problem:

$$u'' + q(u) = f(x), \quad u(a) = \alpha, u(b) = \beta.$$

Approximate the operator $q(u)$ by a linear approximation about the function u_0.

$$q(u) \approx q(u_0) + q'(u_0)(u - u_0),$$

where $q'(u_0)$ denotes the *Fréchet* derivative.

Iterative procedure

First guess a solution for u_0, then solve

$$\begin{aligned} & u_1'' + q'(u_0)(u_1 - u_0) + q(u_0) = f(x), \\ \implies & u_1'' + q'(u_0)u_1 = f(x) - q(u_0) + u_0 q'(u_0), \end{aligned}$$

subject to $u_1(a) = \alpha$ and $u_1(b) = \beta$.

Repeating this process we get

$$\begin{cases} u_{n+1}'' + q'(u_n)u_{n+1} = f(x) - q(u_n) + u_n q'(u_n), \\ u_{n+1}(a) = \alpha, u_{n+1}(b) = \beta, \end{cases}$$

until convergence.

Iterative procedure

Newton update at each iteration

$$\begin{cases} \delta_n'' + q'(u_n)\delta_n = f(x) - q(u_n) - u_n'', \\ u_{n+1} = u_n + \delta_n, \\ \delta_n(a) = 0, \delta_n(b) = 0. \end{cases}$$

For FEM, it is more convenient to first quasi-linearise and then discretise.

End of week 4!