Chapter 1 - Introduction

G. Richardson

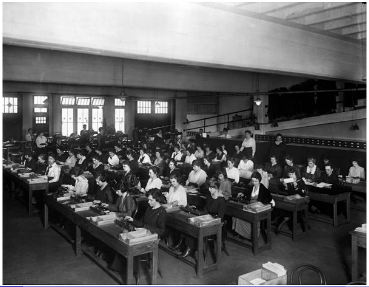
MATH3018/6141, Semester 1

What are numerical methods?

- Algorithmic approach to solving mathematical problems

  Numerical methods are algorithms that give quantitative answers in a finite number of steps.

- Computers are particularly suitable tools for applying these methods because:
  1. problems often require large numbers of computations
  2. computations are repetitive
- Pre 1940's such problems were solved by large teams of people, often with mechanical calculators



Why are numerical methods imporant?

- Very often there is no analytical method to solve a mathematical problem
- Particularly true for *nonlinear* problems, e.g. the simple transcendental equation

  $$\text{Find } x \text{ satisfying} \quad x + e^x = 5\sin x$$

  has no analytical solution.
- Can solve large problems that would otherwise be extremely tedious to solve analytically, e.g.

  $$\text{Solve the matrix equation} \quad A\boldsymbol{x} = \boldsymbol{b}$$

  where A is a square 1000×1000 matrix.
- Can solve linear PDEs for which separation of variables is not appropriate.
- Can solve nonlinear ODEs and PDEs.

Linearity and Nonlinearity

- Nearly all analytical mathematical methods taught at undergraduate level (and below) can only be applied to linear problems
- There are very few *exact* analytic techniques that can be applied to nonlinear problems
- Very few nonlinear problems for which these techniques are applicable
- Faced with a nonlinear problem usually the only recourse is to resort to a numerical method
- A linear operator $\mathcal{L}$ has the property

  $$\mathcal{L}(\alpha \boldsymbol{y} + \beta \boldsymbol{z}) = \alpha \mathcal{L}(\boldsymbol{y}) + \beta \mathcal{L}(\boldsymbol{z}) \quad \text{where } \alpha, \beta \text{ are scalars.}$$

  and $\boldsymbol{y}$ and $\boldsymbol{z}$ could be vectors or funtions or ....
- A linear equation has the form

  $$\text{Find } \boldsymbol{x} \text{ such that } \mathcal{L}\boldsymbol{x} = \boldsymbol{b} \quad \text{where } \boldsymbol{b} \text{ is known}$$

Superposition

- What is special about linear equations of the form

$$\mathcal{L}\boldsymbol{x} = \boldsymbol{b}?$$

- The properties of the linear operator mean that if we can write

$$\mathcal{L}\boldsymbol{x} = \boldsymbol{b} = \sum_{i=1}^{n} B_i \boldsymbol{e}_i$$

  and determine solutions $\boldsymbol{x}_i$ to the problems

$$\mathcal{L}\boldsymbol{x}_i = \boldsymbol{e}_i \qquad \text{for} \quad i = 1, 2, \cdots, n$$

  Then a solution to the original problem is

$$\boldsymbol{x} = \hat{\boldsymbol{x}} = \sum_{i=1}^{n} B_i \boldsymbol{x}_i.$$

- This is the first from of the superposition principle

Superposition (II)

- If $\boldsymbol{x} = \hat{\boldsymbol{x}}$ is a solution to the linear problem

$$\mathcal{L}\boldsymbol{x} = \boldsymbol{b}$$

- And we can find M solutions $\boldsymbol{u}_m$ to the homogenous version of the problem

$$\mathcal{L}\boldsymbol{u} = 0$$

  then, since the operator $\mathcal{L}$ is linear,

$$\boldsymbol{x} = \hat{\boldsymbol{x}} + \sum_{m=1}^{M} \alpha_m \boldsymbol{u}_m$$

  is also a solution to the original problem for any scalars α_m.
- This is the second form of the superposition principle

Using linearity: Inverting a system of linear equations

- Consider the matrix equation

$$A\boldsymbol{x} = \boldsymbol{b} \qquad \text{where } A \text{ is an } n \times n \text{ matrix}$$

- A is a linear operator, since $A(\alpha \boldsymbol{y} + \beta \boldsymbol{z}) = \alpha A\boldsymbol{y} + \beta A\boldsymbol{z}$.
- Use the superposition principle by noting that we can write

$$\boldsymbol{b} = (B_1, B_2, \ldots, B_n)^T$$

in the form

$$\boldsymbol{b} = \sum_{i=1}^{n} B_i \boldsymbol{e}_i \quad \text{where} \quad \boldsymbol{e}_1 = (1, 0, \ldots, 0)^T,$$

$$\boldsymbol{e}_2 = (0, 1, \ldots, 0)^T, \cdots, \boldsymbol{e}_n = (0, 0, \ldots, 1)^T$$

- If we can find the solutions x_i to the n problems

$$A\boldsymbol{x}_i = \boldsymbol{e}_i \qquad \text{for } i = 1, 2, \ldots, n$$

- Then by the superposition principle the solution to $A\boldsymbol{x} = \boldsymbol{b}$ is

$$\boldsymbol{x} = \sum_{i=1}^{n} B_i \boldsymbol{x}_i \quad \Longrightarrow \quad \boldsymbol{x} = A^{-1}\boldsymbol{b} = (\boldsymbol{x}_1 | \boldsymbol{x}_2 | \ldots | \boldsymbol{x}_n)\boldsymbol{b}$$

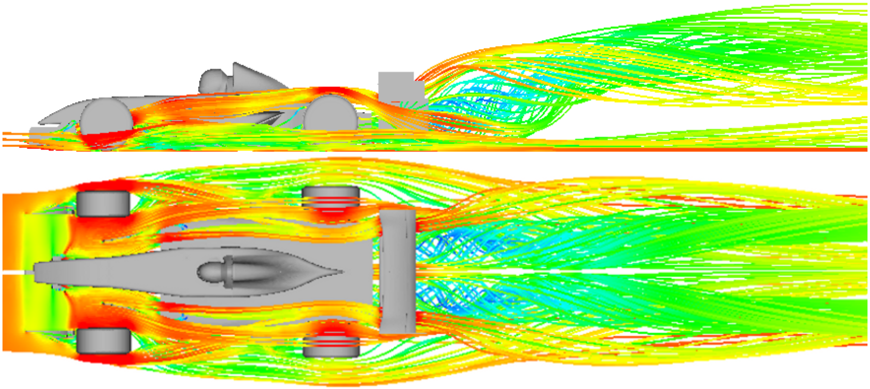
Applying superposition principle to linear ODEs

- The second order ODE
$$\mathcal{L}y = b(x) \quad \text{where} \quad \mathcal{L} = \left(\frac{d^2}{dx^2} + q(x)\frac{d}{dx} + r(x)\right)$$
is linear since $\mathcal{L}$ is a linear differential operator.
- Suppose we can find a particular integral $y = y_{PI}(x)$ that satisfies this equation and two solutions $u_1(x)$ and $u_2(x)$ to the homogenous problem
$$\mathcal{L}u = 0$$
then by the superposition principle
$$y(x) = y_{PI}(x) + \alpha_1 u_1(x) + \alpha_2 u_2(x)$$
is a solution to $\mathcal{L}y = b(x)$.
- Note that we cannot add solutions together to find other solutions if we have a nonlinear ODE, such as
$$\frac{d^2y}{dx^2} = y(1 - y^2)$$
- This makes them much harder to solve.

Remarks on Linear and Nonlinear Problems

- The two forms of the superposition principle underly most of the analytic techniques that you will have come across, *e.g.* inverting matrices, Fourier Series, Fourier Transforms, Laplace Transforms, Separation of Variables etc.
- Such techniques cannot therefore be applied to nonlinear problems
- In such cases normally require a numerical method
- Linear problems are rare. For example the set of all linear 2nd order ODEs is a vanishingly small subset of the set of all 2nd order ODEs
- A mathematician should therefore have some appreciation of numerical methods as they are the only way to solve “most" problems.

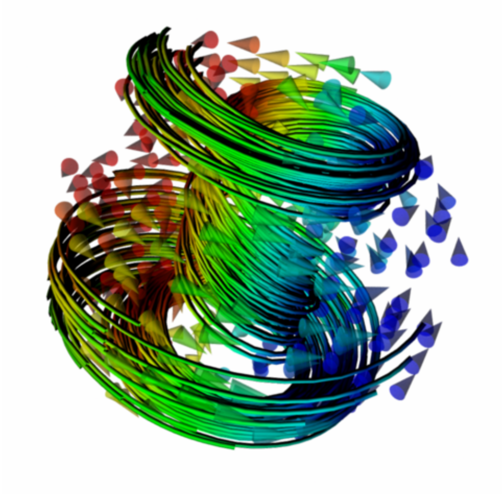
Example of an application of a numerical method (I)



Computation of a coupled set of nonlinear PDEs (the Navier-Stokes Equations) past a complex shape.

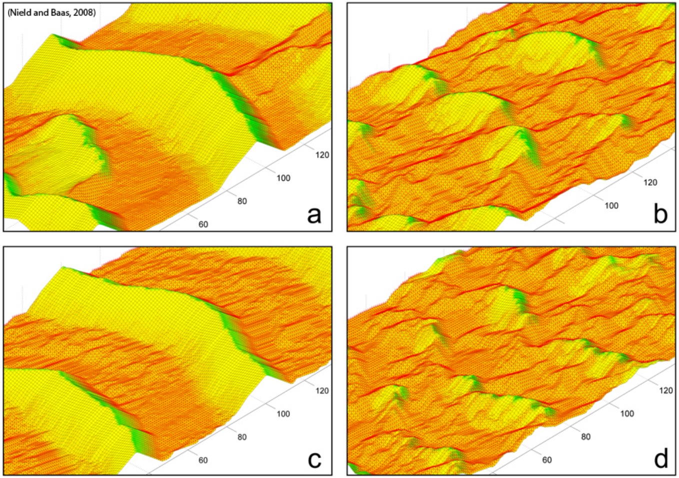
[MSc Race Car Aerodynamics students, School of Engineering Sciences].

Example of an application of a numerical method (II)



Computing solutions to Landau-Lifshitz-Gilbert equations (coupled nonlinear PDEs) to find magnetisation in ferromagnetic particle. [Boardman & Fangohr]

Example of an application of a numerical method (III)



Simulation of sand dune formation using a probabilistic model

[Nield & Baas, Global and Planetary Change 64 (2008)].

What are numerical methods? (2)

Numerical methods are algorithms that give quantitative answers in a finite number of steps.

- If problem is *continuous* (*e.g.* a differential equation) this implies the answer can only be computed approximately. However, with some care, a very accurate numerical approximation to the solution can usually be found.
- The usefulness of algorithm is defined in practice by the number of steps required to obtain an answer to a given accuracy.
- In particular: certain *useless* algorithms exist where *more steps do not improve accuracy*.

Where do numerical methods fit?

Nature / Real World / Experiment

Physics input / assumptions

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0, \qquad (1)$$

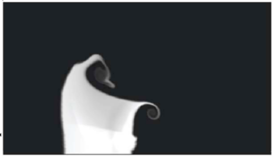
$$\frac{\partial (\rho \mathbf{V}_i)}{\partial t} + \nabla \cdot (\rho \mathbf{V}_i \mathbf{V}) + \frac{\partial p}{\partial \mathbf{x}_i} = \nabla \cdot \tau_i, \qquad (2)$$

$$\frac{\partial E}{\partial t} + \nabla \cdot ((E+p)\mathbf{V}) = \nabla \cdot (\kappa \nabla T) + \nabla \cdot (\underline{\tau}\mathbf{V}), \qquad (3)$$

Model

Numerical Method

Answer



What is this course?

This course will discuss three aspects of numerical methods.

1. How are specific algorithms implemented?
   - "Engineering" aspect.
   - Practical implementation using Python.
   - Class test (10%) and coursework (30%).
   - *You are required to write programs to solve maths problems*
   - *You need to attend the practical labs*
2. How efficient are the algorithms and how can they be implemented efficiently?
   - "Computer science" aspect.
   - Mixture of analytic and practical, Python work.
   - Mainly coursework (30%).
3. How do we derive algorithms and prove that they work?
   - "Mathematics" aspect.
   - Analytic, but feeds into practical work.
   - Closed book exam (60%).

How to think

- Course will *not* introduce cutting edge numerical methods.
- Shows how numerical methods are derived and analyzed.
- The lecture slides define the examinable syllabus and are available on Blackboard. Course notes will be added at a future date.
- Lectures illustrate by example how the algorithms work, can be implemented and used.
- Computer labs introduce the Python needed to illustrate and implement the algorithms.
- Example code for a variety of problems is available on Blackboard.

Finite precision and approximation

Most numerical algorithms involve real numbers. Computer representations in *floating point* notation are approximations:

$$\pi \to 0.\underbrace{314159265358979}_{\text{precision}}\times 10^{\overset{\text{exponent}}{1}}.$$

Precision of representation – number of significant digits – controls accuracy: can never be perfect.

$1+1=\dots$

Imagine a computer with 1 digit of precision. It represents 1 and 0.1 exactly. It also represents

$$1+1=2$$

precisely.
Computer cannot represent sum of 1 and 0.1:

$$\begin{aligned} 1+0.1 &= 1.1 \\ &= 0.11\times 10^1 \\ &\to 0.1\times 10^1 \\ &= 1 \end{aligned}$$

$1+1=2?$

On a computer with a single digit of precision,

$$
\begin{aligned}
1+\sum_{1}^{10} 0.1 &= (1+0.1)+0.1+\ldots 0.1 \\
&= \ldots \\
&= 1
\end{aligned}
$$

or

$$
\begin{aligned}
1+\sum_{1}^{10} 0.1 &= 1+(0.1+\ldots 0.1) \\
&= 1+1 \\
&= 2.
\end{aligned}
$$

Typically Python represents numbers to 16 digit accuracy, *i.e.* better than one part in 10^{16}. Nevertheless accuracy is still sometimes an issue. For example when subtracting two numbers that are very close.

Matrix problems

Many numerical methods depend on solving linear systems. Consider

$$\begin{pmatrix} 4 & 5 \\ 2 & 3 \end{pmatrix} \mathbf{x} = \mathbf{b}.$$

This is a sensible matrix equation.
Problem has solution if (and only if) matrix is non-singular:

$$\det(A) = 4 \times 3 - 5 \times 2 = 2 \neq 0,$$

so problem has a solution. *However*, with only one digit of precision

$$\begin{aligned} \det(A) = 4 \times 3 - 5 \times 2 \qquad & = (12) - (10) \\ & \to (10) - (10) = 0. \end{aligned}$$

Finite precision makes matrix singular; no solution can be found!
With Python would hit trouble if the entries of the matrix are $O(1)$ and its determinant $O(10^{-16})$!

Summary

- Numerical methods are about computing quantitative answers using a finite number of steps.
- The finite accuracy of numerical methods leads to inherent approximations.
- These approximations can lead to fundamental mathematical difficulties.

Chapter 2 - Linear Systems 1

G. Richardson

MATH3018/6141, Semester 1

Introduction

Solving linear "matrix" equations of form

$$A\boldsymbol{x} = \boldsymbol{b}$$

is an important topic in numerical analysis.

Basis for many methods used to solve PDEs and ODE BVPs.

Require efficient and accurate solution of very large linear systems.

Also need to identify situations where our numerical methods are going to struggle to find an accurate solution.

Revisions and reminders

The system of linear equations

$$\begin{cases} x + y = 1 \\ 2x + 2y = 3 \end{cases}$$

has no solutions; the system

$$\begin{cases} 0.9999999x + y = 0.9999999 \\ 2x + 2y = 2.9999999 \end{cases}$$

does.
In general form

$$A\boldsymbol{x} = \boldsymbol{b}$$

has a unique solution iff $\det(A) \neq 0$.

Aim of condition number

The two systems

$$\begin{cases} x + y = 1 \\ 2x + 2y = 3 \end{cases}, \quad \begin{cases} 0.9999999x + y = 0.9999999 \\ 2x + 2y = 2.9999999 \end{cases}$$

are indistinguishable at 6-digit precision; such a “computer” cannot tell if *either* system has a well-defined solution!

Systems are *ill-conditioned* if a “small” input change leads to an enormous change in the solution. The *condition number* should predict ill-conditioned systems without finding solutions first.

Note: as the very notion of “small” depends on the precision, so must the interpretation of the condition number.

Determinants

Think about the determinant $\mathtt{det}(A)$. If it vanishes there is no solution.

But the matrix

$$A = 10^{-1}\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 \end{pmatrix}$$

has determinant 10^{-n}: arbitrarily small, whilst the matrix is trivial to invert.

Determinant $\neq$ condition number.

The error and the condition number

Suppose that we calculate a numerical solution $\tilde{\boldsymbol{x}}$ to the system

$$A\boldsymbol{x} = \boldsymbol{b}$$

the residual $\boldsymbol{r}$ (a measure of the error) to this numerical solution is

$$\boldsymbol{r} = \boldsymbol{b} - A\tilde{\boldsymbol{x}}$$

This cannot be reduced below machine accuracy.

The error $\boldsymbol{e}$ and the relative error ε_{abs} of $\tilde{\boldsymbol{x}}$ are

$$\boldsymbol{e} = \boldsymbol{x} - \tilde{\boldsymbol{x}} \quad \text{and} \quad \varepsilon_{\text{abs}} = \frac{\|\boldsymbol{e}\|}{\|\boldsymbol{x}\|}.$$

How is ε_{abs} related to the residual? It turns out that

$$\varepsilon_{\text{abs}} \leq K(A)\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|} \quad \text{where} \quad K(A) \quad \text{is called condition number.}$$

Very large $K(A)$ means solution cannot be evaluated accurately. We determine an expression for $K(A)$ later.

Vector norms

Norms are “mathematical distance functions”. Focus on real vectors, e.g. to find the error $\|\boldsymbol{x}_{\text{num}} - \boldsymbol{x}_{\text{exact}}\|$. Interesting norms are

$$\|\boldsymbol{x}\|_1 \equiv \sum_{j=1}^{n} |x_j|, \quad \|\boldsymbol{x}\|_2 \equiv \left[\sum_{j=1}^{n} (x_j)^2\right]^{1/2}, \quad \|\boldsymbol{x}\|_\infty \equiv \max_j |x_j|.$$

Example: If $\boldsymbol{x} = (-1, 2, 1)$ then

$$\begin{aligned}
\|\boldsymbol{x}\|_1 &= |-1| + |2| + |1| &&= 4, \\
\|\boldsymbol{x}\|_2 &= \sqrt{(-1)^2 + (2)^2 + (1)^2} &&= \sqrt{6}, \\
\|\boldsymbol{x}\|_\infty &= \max(|-1|, |2|, |1|) &&= 2.
\end{aligned}$$

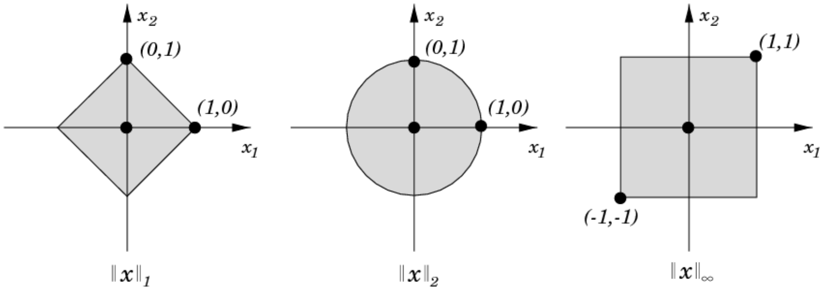
Properties of Vector norms

All vector norms have the following properties

$$\begin{array}{lll}
\|\boldsymbol{x}\| \geq 0 & \text{for all vectors} & \boldsymbol{x}, \\
\|\boldsymbol{x}\| = 0 & \text{only if} & \boldsymbol{x} = \mathbf{0}, \\
\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\| & \text{for all vectors} & \boldsymbol{x}, \boldsymbol{y}, \\
\|c\boldsymbol{x}\| = |c|\|\boldsymbol{x}\| & \text{for all vectors } \boldsymbol{x} & \text{and scalars } c.
\end{array}$$

Can show that $\|\cdot\|_1$, $\|\cdot\|_2$ and $\|\cdot\|_\infty$ norms all satisfy these properties.

Vector norms - geometrically



By plotting the region of $\mathbb{R}^2$ defined by $\|\boldsymbol{x}\| \leq 1$ we see the different geometric meanings of the norms.

Matrix norms

Just as we needed the vector norm to give a measure of the size of a vector we need a measure of the size of a matrix.

The matrix norm provides such a measure for square matrices.

The properties that a matrix norm must have are

$$\begin{array}{ll} \|A\| \geq 0 & \text{for all } n \times n \text{ matrices } A, \\ \|A\| = 0 & \text{only if } A = 0, \\ \|A+B\| \leq \|A\| + \|B\| & \text{for all } n \times n \text{ matrices } A \text{ and } B, \\ \|AB\| \leq \|A\|\|B\| & \text{for all } n \times n \text{ matrices } A \text{ and } B, \\ \|cA\| = |c|\|A\| & \text{for all } n \times n \text{ matrices } A \text{ and scalars } c. \end{array}$$

How are these matrix norms related to vector norms? In particular what do they say about the n-vector $A\boldsymbol{x}$?

A matrix norm $\|\cdot\|_q$ is compatible with vector norm $\|\cdot\|_p$ if

$$\|A\boldsymbol{x}\|_p \leq \|A\|_q \|\boldsymbol{x}\|_p \quad \text{for all } n\text{-vectors } \boldsymbol{x} \ \& \ n \times n\text{-matrices A}$$

Matrix norms

A sensible definition of the matrix $\|\cdot\|_p$ norm is given in terms of the vector $\|\cdot\|_p$ norm by

$$\|A\|_p = \max_{\|\boldsymbol{x}\|\neq 0} \frac{\|A\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p}.$$

This is equivalent to

$$\|A\|_p = \max_{\|\boldsymbol{y}\|=1} \|A\boldsymbol{y}\|_p.$$

In particular it turns out that the $\|\cdot\|_1$ and $\|\cdot\|_\infty$ norms are given by

$$\begin{aligned}
\|A\|_1 &= \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}| = \max(\|\boldsymbol{c}_1\|_1, \|\boldsymbol{c}_2\|_1, \cdots, \|\boldsymbol{c}_n\|_1) \\
\|A\|_\infty &= \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}| = \max(\|\boldsymbol{r}_1\|_1, \|\boldsymbol{r}_2\|_1, \cdots, \|\boldsymbol{r}_n\|_1)
\end{aligned}$$

Important Results for use in Error Analysis

There is one result that we make use of repeatedly when analysing errors in numerical schemes.

This follows from the definition of the matrix $\|\cdot\|_p$ norm that we have used and is

$$\|A\boldsymbol{x}\|_p \leq \|A\|_p \|\boldsymbol{x}\|_p \quad \text{for all } n\text{-vectors } \boldsymbol{x} \text{ \& } n \times n\text{-matrices A}$$

In practice the only norms we make use of through the rest of the

course are the 1- and ∞-norms because they are easy to compute. So the results we need are

$$\|A\boldsymbol{x}\|_1 \leq \|A\|_1 \|\boldsymbol{x}\|_1 \quad \text{for all } n\text{-vectors } \boldsymbol{x} \text{ \& } n \times n\text{-matrices A}$$

and

$$\|A\boldsymbol{x}\|_\infty \leq \|A\|_\infty \|\boldsymbol{x}\|_\infty \quad \text{for all } n\text{-vectors } \boldsymbol{x} \text{ \& } n \times n\text{-matrices A}$$

Examples of matrix norms - 1 norm

The matrix 1-norm reduces to the maximum of the 1-norms of the column vectors of A.

Let

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 4 \\ -1 & 0 & 2 \end{pmatrix}.$$

The column vectors are

$$\boldsymbol{c}_1 = \begin{pmatrix} 1 \\ 4 \\ -1 \end{pmatrix}, \quad \boldsymbol{c}_2 = \begin{pmatrix} 2 \\ 5 \\ 0 \end{pmatrix}, \quad \boldsymbol{c}_3 = \begin{pmatrix} 3 \\ 4 \\ 2 \end{pmatrix}.$$

The resulting 1-norms are

$$\|\boldsymbol{c}_1\|_1 = 6, \quad \|\boldsymbol{c}_2\|_1 = 7, \quad \|\boldsymbol{c}_3\|_1 = 9.$$

Therefore the 1-norm of A is

$$\|A\|_1 = \max(\|\boldsymbol{c}_1\|_1, \|\boldsymbol{c}_2\|_1, \|\boldsymbol{c}_3\|_1) = 9.$$

Examples of matrix norms - ∞ norm

The matrix infinity-norm reduces to the maximum of the 1-norms of the row vectors of A.
Let

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 4 \\ -1 & 0 & 2 \end{pmatrix}$$

as before. The vectors are

$$\boldsymbol{r}_1 = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}, \quad \boldsymbol{r}_2 = \begin{pmatrix} 4 & 5 & 4 \end{pmatrix}, \quad \boldsymbol{r}_3 = \begin{pmatrix} -1 & 0 & 2 \end{pmatrix}.$$

The resulting 1-norms are

$$\|\boldsymbol{r}_1\|_1 = 6, \quad \|\boldsymbol{r}_2\|_1 = 13, \quad \|\boldsymbol{r}_3\|_1 = 3.$$

Therefore the ∞-norm of A is

$$\|A\|_\infty = \max(\|\boldsymbol{r}_1\|_1, \|\boldsymbol{r}_2\|_1, \|\boldsymbol{r}_3\|_1) = 13.$$

Determining the condition number $K(A)$

Choose matrix norm compatible with the vector norm *i.e.* with property

$$\|M\boldsymbol{x}\| \leq \|M\|\|x\| \quad \forall M, \ \ \boldsymbol{x}.$$

Can then deduce the following properties

$$\begin{aligned} \boldsymbol{b} &= A\boldsymbol{x} \Longrightarrow \|\boldsymbol{b}\| \leq \|A\|\|\boldsymbol{x}\| \Longrightarrow \|\boldsymbol{x}\| \geq \|\boldsymbol{b}\|/\|A\| \\ A(\boldsymbol{x}-\tilde{\boldsymbol{x}}) = A\boldsymbol{e} &= \boldsymbol{r} \Longrightarrow \boldsymbol{e} = A^{-1}\boldsymbol{r} \Longrightarrow \|\boldsymbol{e}\| = \|A^{-1}\boldsymbol{e}\| \leq \|A^{-1}\|\|\boldsymbol{r}\| \end{aligned}$$

Write the absolute error ε_{abs} as

$$\varepsilon_{\text{abs}} = \frac{\|\boldsymbol{e}\|}{\|\boldsymbol{x}\|} = \frac{\|A^{-1}\boldsymbol{r}\|}{\|\boldsymbol{x}\|}.$$

and bound it as follows

$$\varepsilon_{\text{abs}} \leq \frac{\|A^{-1}\|\|\boldsymbol{r}\|}{\|\boldsymbol{x}\|} \leq \frac{\|A^{-1}\|\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|/\|A\|} = \|A^{-1}\|\|A\|\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}$$

In other words

$$\varepsilon_{\text{abs}} \leq K(A)\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|} \quad \text{where condition number} \quad K(A) = \|A^{-1}\|\|A\|.$$

Condition number example

Consider the matrix

$$A = \begin{pmatrix} 1 & 2 \\ \frac{499}{1000} & \frac{1001}{1000} \end{pmatrix}.$$

The inverse is

$$A^{-1} = \begin{pmatrix} \frac{1001}{3} & -\frac{2000}{3} \\ -\frac{499}{3} & \frac{1000}{3} \end{pmatrix}.$$

The matrix norms are

$$\|A\|_1 = 3.001, \qquad \|A^{-1}\|_1 = 1000$$
$$\|A\|_\infty = 3, \qquad \|A^{-1}\|_\infty = 1000\tfrac{1}{3}.$$

Hence the condition number is $K(A) = 3001$. Thus if we are only able to work to 3 significant figures then the absolute error may be as high as $\mathcal{O}(1)$ and the result meaningless (see course notes).

Summary

- Linear systems where a “small” relative residual can give rise to a large relative error in the numerical solution are called *ill-conditioned*.
- The *condition number* $K(A) = \|A\|\|A^{-1}\|$ gives a measure for any given matrix without having to compute the solutions.
- Large condition numbers give rise to *ill-conditioned* problems. But whether a condition number is “too large” or not depends on the precision to which you are working and thus how small the residual can be made.
- The condition number depends on matrix norms; for simplicity we only consider simple matrix norms induced by standard vector norms.
- The condition number can be approximated rapidly *without* inverting the matrix; this makes it useful as a pre-check.

Chapter 3 - Solving Linear Systems

G. Richardson

MATH3018/6141, Semester 1

Direct vs. Indirect Methods

We want to solve the linear system

$$A\boldsymbol{x} = \boldsymbol{b}$$

for (reasonably conditioned) arbitrary A. Can either use a

- Direct Method: a finite algorithm. Obtains answer in a fixed number of steps,
- Indirect (or Iterative) Method: an infinite algorithm. Converges toward the answer.

Classic direct method is Gaussian Elimination. But have met this in Linear Algebra Courses and not often used in practice so not treated here.

Direct is *not* always best:

1. For a large matrix ($n \gtrsim 10^5$) a direct method may be impractical.
2. Accumulation of round-off error may make the direct method considerably less accurate than the indirect method.

Decomposition Methods

We (still) want to solve the linear system

$$A\boldsymbol{x} = \boldsymbol{b}.$$

One approach is to split A into the product of matrices, then solve the succession of (easier) linear systems. For example, set

$$A = LU,$$

solve the problem

$$L\boldsymbol{y} = \boldsymbol{b},$$

and then solve the problem

$$U\boldsymbol{x} = \boldsymbol{y}$$

for the original unknown $\boldsymbol{x}$.
Only makes sense if new linear systems are straightforward to solve.
Here we only consider the LU decomposition, where L and U are lower and upper triangular matrices.

Example

$$\begin{pmatrix} 2 & 1 & -1 \\ 4 & 1 & 0 \\ -2 & -3 & 8 \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 0 \\ 6 \\ -12 \end{pmatrix}$$

will be solved by writing $A = LU$ and then using forward and backward substitution to solve the sub-problems.

Example

The matrix can be decomposed as

$$\begin{pmatrix} 2 & 1 & -1 \\ 4 & 1 & 0 \\ -2 & -3 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & -1 \\ 0 & -1 & 2 \\ 0 & 0 & 3 \end{pmatrix}.$$

Then solve $L\boldsymbol{y} = \boldsymbol{b}$ by *forward* substitution:

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 2 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \\ -12 \end{pmatrix} \Rightarrow \begin{cases} y_1 = 0, \\ y_2 = 6, \\ y_3 = -24. \end{cases}$$

Example

The matrix can be decomposed as

$$\begin{pmatrix} 2 & 1 & -1 \\ 4 & 1 & 0 \\ -2 & -3 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & -1 \\ 0 & -1 & 2 \\ 0 & 0 & 3 \end{pmatrix}.$$

Then solve $L\boldsymbol{y} = \boldsymbol{b}$ by *forward* substitution:

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 2 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \\ -12 \end{pmatrix} \Rightarrow \begin{cases} y_1 = 0, \\ y_2 = 6, \\ y_3 = -24. \end{cases}$$

Then solve $U\boldsymbol{x} = \boldsymbol{y}$ by *backward* substitution:

$$\begin{pmatrix} 2 & 1 & -1 \\ 0 & -1 & 2 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \\ -24 \end{pmatrix} \Rightarrow \begin{cases} x_1 = 7, \\ x_2 = -22, \\ x_3 = -8. \end{cases}$$

How do we factorise?

The factorisation is not unique: L and U together have $n^2 + n$ free coefficients; A only has n^2. Can freely choose n coefficients. Two obvious choices are

1. Doolittle's factorisation: The diagonal entries of L are 1.
2. Crout's factorisation: The diagonal entries of U are 1.

Then use the explicit matrix multiplication formula, and work from the top left corner. Each coefficient obeys

$$a_{ij} = \sum_{s=1}^{n} \ell_{is} u_{sj} = \sum_{s=1}^{\min(i,j)} \ell_{is} u_{sj}$$

where the last equality holds because L and U are triangular: $\ell_{is} = 0$ for $s > i$ and $u_{sj} = 0$ for $s > j$.

Factorisation by example

The previous example had the matrix

$$A = \begin{pmatrix} 2 & 1 & -1 \\ 4 & 1 & 0 \\ -2 & -3 & 8 \end{pmatrix}$$

which we decompose using

$$a_{ij} = \sum_{s=1}^{\min(i,j)} \ell_{is} u_{sj}.$$

Here we factorise with the choice $\ell_{11} = \ell_{22} = \ell_{33} = 1$.

Then the first coefficient, $i = j = 1$, obeys

$$a_{11} = \ell_{11} u_{11} \implies u_{11} = a_{11} = 2.$$

Factorisation by example - II

Now consider the first row of U ($i = 1$, j free) and the first column of L (i free, $j = 1$). The first row of U obeys

$$a_{1j} = \ell_{11} u_{1j} \Longrightarrow u_{1j} = a_{1j} \quad \text{for} \;\; j = 1, 2, 3$$

and the first column of L obeys

$$a_{i1} = \ell_{i1} u_{11} \Longrightarrow \ell_{i1} = a_{i1}/u_{11} = a_{i1}/2 \quad \text{for} \;\; i = 1, 2, 3.$$

Therefore we know that

$$L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{4}{2} & ?? & 0 \\ \frac{-2}{2} & ?? & ?? \end{pmatrix}, \quad U = \begin{pmatrix} 2 & 1 & -1 \\ 0 & ?? & ?? \\ 0 & 0 & ?? \end{pmatrix}.$$

Factorisation by example - III

Go to the second row.

$$\begin{aligned} a_{2j} &= \ell_{21} u_{1j} + \ell_{22} u_{2j} \quad \text{for } \; j = 2, 3 \\ \Longrightarrow u_{2j} &= a_{2j} - \ell_{21} u_{1j} \quad \text{for } \; j = 2, 3 \end{aligned}$$

Already computed entries $\ell_{21} = 2$, $u_{12} = 1$, $u_{13} = -1$. It follows that

$$\begin{aligned} u_{22} &= 1 - 2 \times 1 = -1 \\ u_{23} &= 0 - 2 \times (-1) = 2. \end{aligned}$$

From the second column we find

$$\begin{aligned} a_{i2} &= \ell_{i1} u_{12} + \ell_{i2} u_{22} \quad \text{for } \; i = {\color{red}2}, 3, \\ \Longrightarrow \ell_{i2} &= (a_{i2} - \ell_{i1} u_{12}) / u_{22} \quad \text{for } \; i = {\color{red}2}, 3, \end{aligned}$$

which implies that

$$\ell_{32} = (-3 - (-1) \times 1) / (-1) = 2.$$

Factorisation by example - IV

We now have the first two rows and columns:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 2 & ?? \end{pmatrix}, \quad U = \begin{pmatrix} 2 & 1 & -1 \\ 0 & -1 & 2 \\ 0 & 0 & ?? \end{pmatrix}.$$

Continuing as above to the third row

$$\begin{aligned} a_{33} &= \ell_{31} u_{13} + \ell_{32} u_{23} + \ell_{33} u_{33} \\ \Longrightarrow u_{33} &= (a_{33} - \ell_{31} u_{13} - \ell_{32} u_{23})/\ell_{33} = (a_{33} - \ell_{31} u_{13} - \ell_{32} u_{23}) \\ \Longrightarrow u_{33} &= 8 - (-1) \times (-1) - 2 \times 2 = 3 \end{aligned}$$

So finally we have

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 2 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 2 & 1 & -1 \\ 0 & -1 & 2 \\ 0 & 0 & 3 \end{pmatrix}.$$

Relation to Gaussian Elimination

There is a close relation between LU decomposition and Gaussian Elimination.

In fact the first set of operations used in Gaussian Elimination reduce the matrix to upper triangular form. These consist of multiplying matrix A by a sequence of lower triangular matrices $L_1, L_2, \dots, L_{n-1}$ to give an upper triangular matix U.

$$L_{n-1}L_{n-2}\dots L_2L_1A = U$$

This transforms the equation

$$A\boldsymbol{x} = \boldsymbol{b} \quad \Longrightarrow \quad U\boldsymbol{x} = L_{n-1}L_{n-2}\dots L_2L_1\boldsymbol{b}$$

Notably a product of lower triangular matrices gives a lower triangular matrix AND the inverse of this lower triangular matrix gives another lower triangular matrix, say L. This suggest writing

$$L = (L_{n-1}L_{n-2}\dots L_2L_1)^{-1} \quad \text{and implies} \quad A = LU$$

Relation to Gaussian Elimination II

What are the lower trinagular matrices L_1, L_2, $\ldots$, L_{n-1} used in Gaussian elimination?

$L_1 A =$

$$\begin{pmatrix} 1 & 0 & 0 & \ldots \\ -\frac{a_{21}}{a_{11}} & 1 & 0 & \ldots \\ -\frac{a_{31}}{a_{11}} & 0 & 1 & \ldots \\ -\frac{a_{41}}{a_{11}} & 0 & 0 & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots \\ a_{21} & a_{22} & a_{23} & \ldots \\ a_{31} & a_{32} & a_{33} & \ldots \\ a_{41} & a_{42} & a_{43} & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots \\ 0 & b_{22} & b_{23} & \ldots \\ 0 & b_{32} & b_{33} & \ldots \\ 0 & b_{42} & b_{43} & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{pmatrix}$$

$L_2(L_1 A) =$

$$\begin{pmatrix} 1 & 0 & 0 & \ldots \\ 0 & 1 & 0 & \ldots \\ 0 & -\frac{b_{32}}{b_{22}} & 1 & \ldots \\ 0 & -\frac{b_{42}}{b_{22}} & 0 & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots \\ 0 & b_{22} & b_{23} & \ldots \\ 0 & b_{32} & b_{33} & \ldots \\ 0 & b_{42} & b_{43} & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots \\ 0 & b_{22} & b_{23} & \ldots \\ 0 & 0 & c_{33} & \ldots \\ 0 & 0 & c_{43} & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{pmatrix}$$

Relation to Gaussian Elimination III

How do we compute L in the decomposition $A = LU$? We know

$$L = (L_{n-1}L_{n-2}\dots L_2L_1)^{-1} = L_1^{-1}L_2^{-1}\dots L_{n-2}^{-1}L_{n-1}^{-1}$$

It is easy to invert the matrices L_1, L_2, etc. Just need to put a minus sign in front of the subdaigonal entries. Hence

$$L_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & \dots \\ \frac{a_{21}}{a_{11}} & 1 & 0 & \dots \\ \frac{a_{31}}{a_{11}} & 0 & 1 & \dots \\ \frac{a_{41}}{a_{11}} & 0 & 0 & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix}, \quad L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & \dots \\ 0 & 1 & 0 & \dots \\ 0 & \frac{b_{32}}{b_{22}} & 1 & \dots \\ 0 & \frac{b_{42}}{b_{22}} & 0 & \dots \\ \dots & \dots & \dots & \dots, \end{pmatrix} \quad \text{etc.}$$

Pivoting

- LU decomposition fails if any (except the last) diagonal component of U is zero because we divide by u_{jj} (for $j = 1, 2, \dots, n-1$) to find other coefficients.
- For large matrices process can be unstable, even where none of $u_{jj} = 0$, because repeated division by small coefficients can lead to large inaccurate entries in L and U.
- For Gaussian elimination the problem is avoided by pivoting (*i.e.* swapping rows of the matrix) when a diagonal entry is small. Equivalent to permuting rows of matrix equation $A\boldsymbol{x} = \boldsymbol{b}$. And accomplished by multiplying by a permutation matrix P, *i.e.* $PA\boldsymbol{x} = P\boldsymbol{b}$. E.G.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

swaps the second and third rows (equivalently swaps the order of the 2nd and 3rd equations).

Summary of the algorithm

- Write down the explicit formula for the matrix coefficients.
  1. Work from the first row / column to the last.
  2. Look at the diagonal entry of A: use the freedom to choose the value of either L or U's diagonal entry.
  3. The appropriate row of U and column of L follows from the explicit formula as all other entries are known.
- If either u_{kk} or ℓ_{kk} are zero the simple algorithm fails; however, an LU decomposition may still exist. Try pivoting.
- U is the matrix that would be found using Gaussian elimination. LU decomposition has the advantage that it holds for any $\boldsymbol{b}$ in $A\boldsymbol{x} = \boldsymbol{b}$, whereas with Gaussian elimination the whole algorithm needs repeating.
- As with Gaussian elimination, pivoting for accuracy is necessary for general matrices.

Conditions for factorisation - background

We *assumed* an LU decomposition is always possible. However, we then noted that the algorithm works only if both u_{kk} and ℓ_{kk} are non-zero, which is not known in advance.

Two key concepts used in determining if a factorisation exists.

1. The *principal minor* of order k of a matrix A is the (sub-) matrix
$$\begin{pmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{k1} & \cdots & a_{kk} \end{pmatrix}.$$
2. A matrix is *strictly diagonally dominant* if
$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \quad (1 \leq i \leq n).$$

Conditions for factorisation - theorems

Summarize the two key theorems; see references given in the notes:

Theorem 1.1: If all $n-1$ leading principal minors of the $n \times n$ matrix A are nonsingular, then an LU decomposition of A exists.

The relation between LU decomposition and Gaussian elimination is exploited to prove this theorem. It also shows that, if the matrix is nonsingular, its rows can be permuted so that an LU decomposition of the permuted matrix can be found.

Theorem 1.2: Every strictly diagonally dominant matrix is nonsingular and has an LU decomposition.

Diagonal dominance is a very simple condition to check. Most matrices involved in numerical methods for solving PDEs are diagonally dominant, leading to the practical utility of this theorem.

Summary

- Direct Methods solve linear systems in finite number of steps. Indirect (or Iterative) Methods are infinite but can be truncated to give an approximate solution.
- Indirect Methods are often faster to reach given accuracy; accumulated round-off error mean direct methods are not “exact”.
- Decomposition methods rewrite the matrix A as product of two matrices resulting in two sub-problems that are easy to solve.
- LU decomposition is the simplest example.
  - The sub-problems are triangular matrix problems, solved by forwards or backwards substitution.
  - There are n free choices in the coefficients of L and/or U, which can be picked to simplify the method.
  - Algorithm may not work even when an LU decomposition exists.
  - LU decomposition is closely related to Gaussian elimination, but is independent of $\boldsymbol{b}$ and hence more efficient.
  - A non-singular matrix can be permuted to one with LU decomposition (diagonally dominant matrices have an LU decomposition).

Chapter 4 - Iterative Methods

G. Richardson

MATH3018/6141, Semester 1

Iterative Methods

Can use an indirect Iterative Mehtod to solve the linear system

$$A\boldsymbol{x} = \boldsymbol{b}.$$

All direct methods (*e.g.* LU decomposition) require $\mathcal{O}(n^3)$ operations. For large matrices this takes time and each operation introduces error that may accumulate. Other methods may be preferable.

Iterative methods build a *sequence* of approximate solutions. Start from “guess” $\boldsymbol{x}^{(0)}$ and compute successive approximations $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}$ which converge to the exact solution.

Sequence can be truncated when sufficiently accurate solution found.

Key elements of an iterative method:

1. Rapid convergence to the correct solution,
2. Fast algorithm for computing each successive approximation.

A general framework

Standard “notation” is to split the matrix as

$$A = N - P,$$

where N and P are (as yet unknown) $n \times n$ matrices. The linear system becomes

$$N\boldsymbol{x} = P\boldsymbol{x} + \boldsymbol{b}.$$

Then *assume* that we have an approximate solution, or guess, $\boldsymbol{x}^{(0)}$, and define the sequence

$$N\boldsymbol{x}^{(i)} = P\boldsymbol{x}^{(i-1)} + \boldsymbol{b}, \quad i = 1, 2, \ldots$$

Obviously consistent with the original equation. The algorithm will only be useful if

1. N is nonsingular, and
2. The linear system $N\boldsymbol{y} = \boldsymbol{z}$ is “easy” to solve.

Convergence

To work, the sequence

$$\boldsymbol{x}^{(i)} = N^{-1}\left(P\boldsymbol{x}^{(i-1)} + \boldsymbol{b}\right), \quad i = 1, 2, \dots$$

must have a limit. To investigate the convergence we note that

$$\boldsymbol{b} = N\boldsymbol{x} - P\boldsymbol{x} \Longrightarrow N^{-1}\boldsymbol{b} = \boldsymbol{x} - N^{-1}P\boldsymbol{x}$$

Substitute this into the sequence

$$\begin{aligned} \boldsymbol{x}^{(i)} &= N^{-1}P\boldsymbol{x}^{(i-1)} + N^{-1}\boldsymbol{b} = N^{-1}P(\boldsymbol{x}^{(i-1)} - \boldsymbol{x}) + \boldsymbol{x} \\ \Longrightarrow \boldsymbol{e}^{(i)} &= N^{-1}P\boldsymbol{e}^{(i-1)} \quad \text{where } \boldsymbol{e}^{(i)} = \boldsymbol{x} - \boldsymbol{x}^{(i)}. \end{aligned}$$

Since we wish error $\boldsymbol{e}^{(i)}$ to decay it is clear that the properties of matrix $M = N^{-1}P$ are key to the convergence. In particular we note

$$\|\boldsymbol{e}^{(i)}\| = \|M\boldsymbol{e}^{(i-1)}\| \leq \|M\|\|\boldsymbol{e}^{(i-1)}\|$$

And so the sequence will converge

$$\|M\| = \|N^{-1}P\| < 1.$$

Convergence (II)

The result

$$\|\boldsymbol{e}^{(n)}\| \leq \|M\|\|\boldsymbol{e}^{(n-1)}\|$$

is easily extended to

$$\|\boldsymbol{e}^{(n)}\| \leq \|M\|^n\|\boldsymbol{e}^{(0)}\|$$

When plotting error it is more useful to make log plot in which we plot $\log_e(\|\boldsymbol{e}^{(n)}\|)$ against n. Then find that

$$\log_e(\|\boldsymbol{e}^{(n)}\|) \leq -n \log_e\left(\frac{1}{\|M\|}\right) + \log_e(\|\boldsymbol{e}^{(0)}\|)$$

I.E. log error lies below a straight line with slope $-\log_e\left(\frac{1}{\|M\|}\right)$.

General assumptions

In what follows we shall always assume that all diagonal entries are 1.

If this is not true, we can always arrange it to be so by dividing each row by its diagonal element.

If the diagonal element of any row is zero, permute the rows. It must be possible to arrange a *nonsingular* matrix such that every diagonal element is non-zero just by row operations.

Jacobi's Method

One key requirement on the split

$$A = N - P$$

was that the $N\boldsymbol{y} = \boldsymbol{z}$ should be easy to solve. The simplest possible problem would be $N = I$.

This means (as A is 1 on the diagonal) that

$$P = A_L + A_U$$

where A_L and A_U are the "triangular" parts of $-A$ (note sign!). The iteration scheme is

$$\boldsymbol{x}^{(i)} = (A_L + A_U)\boldsymbol{x}^{(i-1)} + \boldsymbol{b}, \quad i = 1, 2, \ldots$$

and the convergence matrix is $M = P$.

Example of Jacobi's Method

We look at the solution of the simple linear system

$$\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \end{pmatrix}$$

using Jacobi's Method. Ensure the diagonal elements are all 1:

$$A = \begin{pmatrix} 1 & \frac{1}{3} \\ \frac{1}{3} & 1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} \frac{5}{3} \\ \frac{7}{3} \end{pmatrix}.$$

Check convergence by computing

$$M = N^{-1}P = P \begin{pmatrix} 0 & -\frac{1}{3} \\ -\frac{1}{3} & 0 \end{pmatrix}$$

To check for convergence

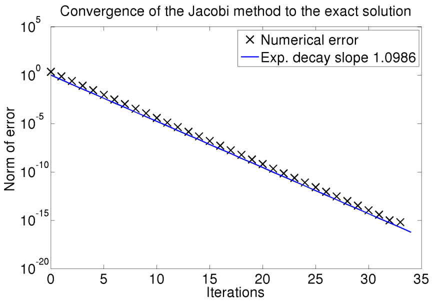
$$\|M\|_1 = \max(\|\boldsymbol{c}\|_1, \|\boldsymbol{c}\|_2) = \max(1/3, 1/3) = 1/3 \Longrightarrow \|M\|_1 < 1.$$

which implies that the method converges.

Example of Jacobi’s Method 2

The sequence has entries

i	x_1	x_2
0	0	0
1	$5/3$	$7/3$
2	0.888889	1.777778
5	1.008230	2.004115
10	0.999983	1.999966
100	1.000000	2.000000



A starting “guess” of $\boldsymbol{x} = \mathbf{0}$ appears to converge to $\boldsymbol{x} = (1, 2)^T$. The convergence is exponential with a slope ~ -1.1. Notice that $\log_e(\|M\|_1) = \log_e(1/3) = -1.0986$.

Gauss-Seidel Method

In the Gauss-Seidel method split the coefficient matrix as

$$A = I - (A_L + A_U)$$

as before. Choose

$$\begin{aligned} N &= I - A_L, \\ P &= A_U. \end{aligned}$$

This makes the iteration scheme

$$\boldsymbol{x}^{(i)} = A_L \boldsymbol{x}^{(i)} + A_U \boldsymbol{x}^{(i-1)} + \boldsymbol{b}, \quad i = 1, 2, \ldots$$

Written in this form as A_L *strictly* lower triangular (*i.e.* all elements above the diagonal *and on the diagonal* are zero). Consider each component of the iteration scheme from the top down: each coefficient of $\boldsymbol{x}^{(i)}$ is computed on the left before it is used on the right.

Example of Gauss-Seidel Method

As above we look at

$$\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \end{pmatrix} \to \begin{pmatrix} 1 & \frac{1}{3} \\ \frac{1}{3} & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \frac{5}{3} \\ \frac{7}{3} \end{pmatrix}.$$

We have

$$A_L = \begin{pmatrix} 0 & 0 \\ -1/3 & 0 \end{pmatrix}, \quad A_U = \begin{pmatrix} 0 & -1/3 \\ 0 & 0 \end{pmatrix}$$

and hence the iteration algorithm becomes

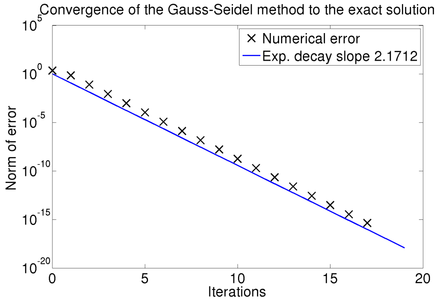
$$\begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ -1/3 & 0 \end{pmatrix} \begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \end{pmatrix} + \begin{pmatrix} 0 & -1/3 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1^{(i-1)} \\ x_2^{(i-1)} \end{pmatrix} + \begin{pmatrix} \frac{5}{3} \\ \frac{7}{3} \end{pmatrix}$$
$$= \frac{1}{3} \begin{pmatrix} 5 - x_2^{(i-1)} \\ 7 - x_1^{(i)} \end{pmatrix}.$$

If read row-by-row from the top we have always computed the $\boldsymbol{x}^{(i)}$ entries before they are used.

Example of Gauss-Seidel Method 2

The sequence has entries

i	x_1	x_2
0	0	0
1	$5/3$	$16/9$
2	1.074074	1.975309
5	1.000102	1.999966
10	1.000000	2.000000
100	1.000000	2.000000



A starting "guess" of $\boldsymbol{x} = \mathbf{0}$ appears to converge to $\boldsymbol{x} = (1, 2)^T$. The convergence is exponential with a slope ~ 2, much faster than Jacobi's Method.

Successive Over Relaxation

View the sequence as repeated “corrections” to the previous term,

$$\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i-1)} + \boldsymbol{c}^{(i)},$$

where correction is based e.g. on Gauss-Seidel method,

$$\boldsymbol{c}^{(i)} = A_L \boldsymbol{x}^{(i)} + (A_U - I)\boldsymbol{x}^{(i-1)} + \boldsymbol{b}.$$

Modify correction by factor ω:

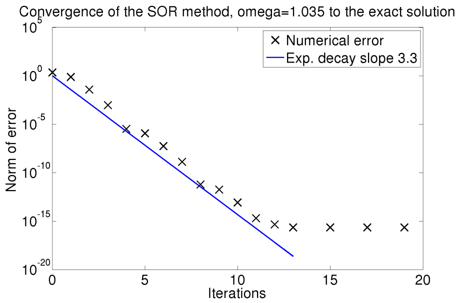
$$\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i-1)} + \omega \boldsymbol{c}^{(i)}.$$

A priori best value of ω not clear. Typically it is set close to one. However, poor choices may cause the algorithm to diverge!

Example of SOR Method 1

Solving the previous example using $\omega = 1.035$, the sequence has entries

i	x_1	x_2
0	0	0
1	1.725000	1.819875
2	1.036768	1.993619
5	0.999999	2.000000
10	1.000000	2.000000
100	1.000000	2.000000

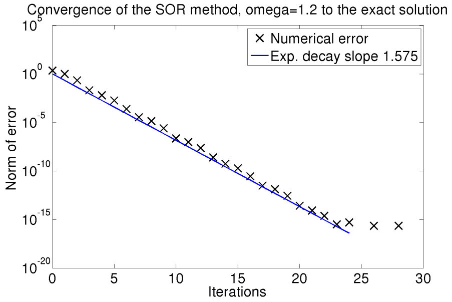


A starting "guess" of $\boldsymbol{x} = \mathbf{0}$ appears to converge to $\boldsymbol{x} = (1, 2)^T$. The convergence is exponential with a slope ~ 3, the fastest method so far.

Example of SOR Method 2

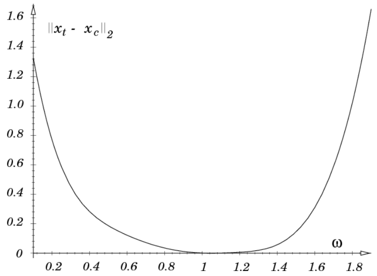
Solving the previous example using $\omega = 1.2$, the sequence has entries

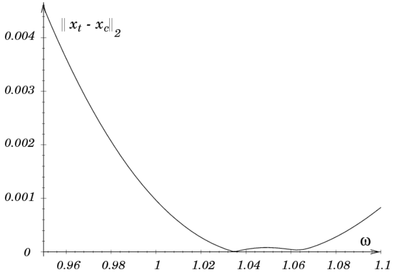
i	x_1	x_2
0	0	0
1	2.000000	2.000000
2	0.800000	2.080000
5	0.998221	2.000430
10	1.000000	2.000000
100	1.000000	2.000000



A starting "guess" of $\boldsymbol{x} = \mathbf{0}$ appears to converge to $\boldsymbol{x} = (1, 2)^T$. The convergence is exponential with a slope ~ 1.6, which is not as good as Gauss-Seidel.

The relaxation parameter





The norm of the error obtained with SOR after four iterations. The convergence behaviour depends in a highly non-trivial way on ω.

SOR algorithms are much harder to analyze than Jacobi or Gauss-Seidel.

Convergence analysis

Two theorems about *when* these methods will work:

Theorem 1: If the coefficient matrix A is *strictly* diagonally dominant then both the Jacobi method and the Gauss-Seidel method will converge.

Recall that a matrix is *strictly diagonally dominant* if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \quad (1 \leq i \leq n)$$

Theorem 2: If the coefficient matrix A is positive definite (symmetric with positive eigenvalues) then the Gauss-Seidel method will converge.

For a real matrix to be positive definite it must be symmetric and all of its eigenvalues (which are therefore real) must also be positive.

Other matrix operations

We have so far only considered the linear system

$$A\boldsymbol{x} = \boldsymbol{b}.$$

We have given methods to solve this system accurately without computing the matrix inverse or determinant. However, these methods can also be used to efficiently and accurately compute the following matrix properties.

Determinants

Given the *LU* decomposition we have

$$\det(A) = \det(L) \times \det(U), \quad \text{and} \quad \det(U) = \prod_{i=1}^{n} u_{ii}.$$

Choose $\ell_{ii} = 1$ giving $\det(L) = \prod_{i=1}^{n} \ell_{ii} = 1$ and the determinant of A follows.

Note: LU decomposition, and hence finding the determinant, is an $\mathcal{O}(n^3)$ operation. The expansion in minors is an $\mathcal{O}(n!)$ operation.

Matrix inversion

Solving for the matrix inverse can be written as a set of linear system problems. We write

$$AA^{-1} = I$$

and consider each column of A^{-1} as a separate problem. Writing $\boldsymbol{c}_i$ for the (unknown) column of A^{-1} we have

$$A^{-1} = \left(\boldsymbol{c}_1 | \boldsymbol{c}_2 | \cdots | \boldsymbol{c}_n\right) \qquad \text{and} \qquad A\boldsymbol{c}_i = \begin{pmatrix} \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix}$$

Here the 1 appears in the i^{th} row of RHS.

This finds the inverse by solving n linear systems, which is much faster than evaluating the $n+1$ determinants required by Cramer's rule.

Summary

- Iterative methods are often based on the split

$$A = N - P.$$

- Convergence depends on $M = N^{-1}P$.
- Jacobi chooses $N = I$ and converges slowly.
- Gauss-Seidel chooses $N = I - A_U$ and uses coefficients as soon as they become available. Slightly harder to code, it converges faster than Jacobi.
- SOR tries to “accelerate” the convergence of Gauss-Seidel depending on a parameter ω. It can be faster, but the behaviour depends non-trivially on ω and it is very difficult to analyze.
- Strict diagonal dominance is sufficient for convergence of both Jacobi and Gauss-Seidel methods.
- Computing determinants and inverses from the solution of linear systems is usually much faster than “standard” techniques.

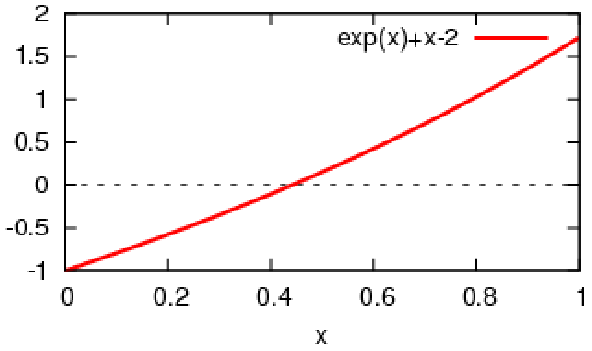
Chapter 5 - Nonlinear equations and bisection

G. Richardson

MATH3018/6141, Semester 1

Nonlinear equations

A large number of nonlinear equations obviously *have* solutions, but the solutions cannot always be found in closed form.
Consider solving the nonlinear equation $\exp(x) + x = 2$. No closed form analytic solution.



Numerical methods can give accurate, approximate solutions.

Framework

Write the problem as

$$f(x) = 0.$$

Goal: given f, find x.

Assume: f real, continuous. Sometimes restrict to f differentiable.

May consider equivalent problem

$$g(x) = x.$$

Same problem, different (geometric) interpretation.

Assume that evaluating f or g is *expensive*. Therefore want to minimize the number of these function evaluations.

Bisection method

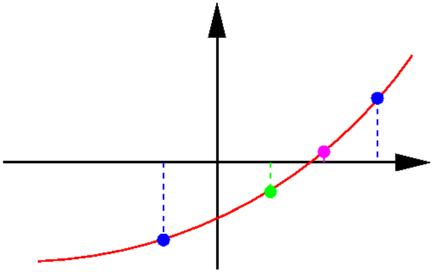
Given $f : x \in \mathbb{R} \to \mathbb{R}$, continuous, find x such that

$$f(x) = 0.$$

Simple and robust method: *bisection*. Assume we have found (somehow!) two points, $x^{(L)}$ and $x^{(R)}$ that bracket root.

Check halfway point $x^{(M)}$. Either $f(x^{(M)}) = 0$ (problem solved), or $x^{(M)}$ and one of $x^{(L,R)}$ bracket the root.

Repeat to converge.

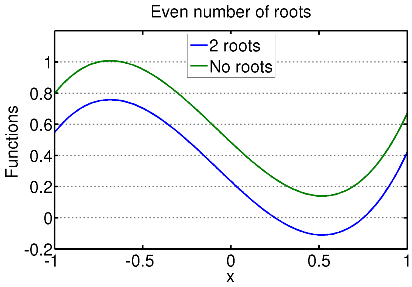


Starting

To start need at least one root in $[x^{(L)}, x^{(R)}]$. Can compute $f(x^{(L)})$, $f(x^{(R)})$.

Simple test $f(x^{(L)}) \times f(x^{(R)}) < 0$ works for odd number of roots.

Cannot distinguish even number of roots and no roots. Doubling size of interval may work; otherwise human intervention best.

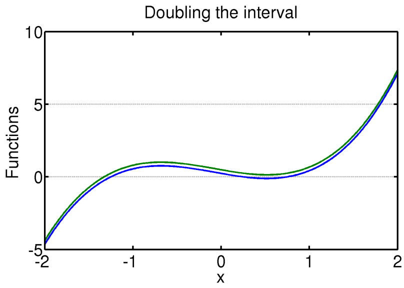


Starting

To start need at least one root in $[x^{(L)}, x^{(R)}]$. Can compute $f(x^{(L)})$, $f(x^{(R)})$.

Simple test $f(x^{(L)}) \times f(x^{(R)}) < 0$ works for odd number of roots.

Cannot distinguish even number of roots and no roots. Doubling size of interval may work; otherwise human intervention best.

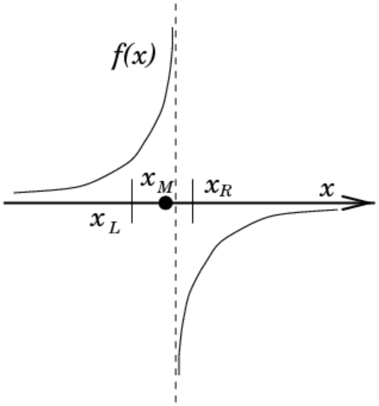


Stopping

When is the answer "accurate enough"? Two obvious stopping criteria:

1. $|f(x_n^{(M)})| < \epsilon$.

This fails in the case on the right.
In fact there is no solution!



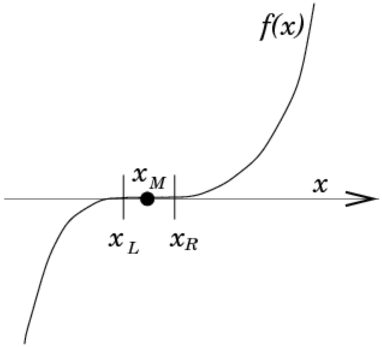
Stopping

When is the answer “accurate enough”? Two obvious stopping criteria:

1. $|f(x_n^{(M)})| < \epsilon$.
2. $|x_n^{(R)} - x_n^{(L)}| < \delta$.

This fails in the case on the right.

Use both, and restrict number of iterations.



Example

Apply bisection to

$$f(x) = e^x + x - 2 = 0$$

on the domain $x \in [0, 1]$.

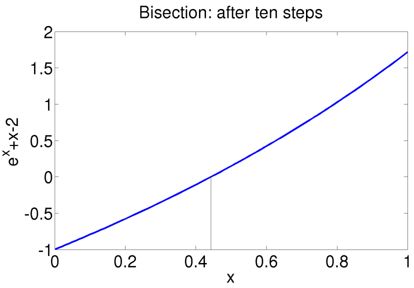
Start: interval is entire domain; function values show an odd number of roots.

Two steps: 25% of interval.

Five steps: 3% of interval.

Ten steps: root “found” to good approximation.

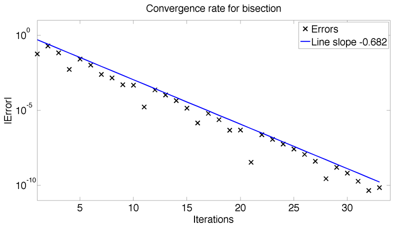
33 steps: function is zero to 10^{-10}.



Bisection convergence

Convergence is *linear*: each iteration reduces error by constant factor.

Clear as root inside interval; error reduces by constant amount each iteration.



Summary

- The simplest method for solving nonlinear equations $f(x) = 0$ is the bisection method:
  - Find two points $x^{(L)}$, $x^{(R)}$ such that $f(x^{(L)})\, f(x^{(R)}) < 0$.
  - Find the midpoint $x^{(M)}$. If this is not sufficiently accurate, replace either $x^{(L)}$ or $x^{(R)}$ with $x^{(M)}$ as appropriate so that the root is still bracketed.
  - Repeat as necessary.
- Normally to stop the algorithm any one of the following is sufficient:
  1. Too many iterations are performed.
  2. The estimated root $f(x^{(M)})$ is sufficiently small.
  3. The interval is sufficiently small.
- The error in the root location at step n is always less then $(x_0^{(R)} - x_0^{(L)})/2^n$.
- Bisection always converges, but is slow.
- Generalizing to systems is hard; how to bracket root?

Chapter 6 - Functional Iteration Methods

G. Richardson

MATH3018/6141, Semester 1

Iterative methods for nonlinear equations

Want to find root of nonlinear equation

$$f(x) = 0.$$

Class of *iterative* methods start from guess x_0, construct sequence

$$x_0, x_1, x_2, \dots, x_n$$

converging to s such that $f(s) = 0$.

Define $g(x)$ in terms of $f(x)$ so that $\quad f(x) = 0 \Longleftrightarrow x = g(x).$

Then introduce the interative scheme

$$x_{n+1} = g(x_n).$$

Key requirements for the method are

- The sequence constructed by g converges to a limit (s);
- The limit constructed is a root of f.

Functional iteration methods

Simplest methods: if limit of sequence exists, it must be fixed point of g: $g(s) = s$.

Thus, if s exists, we have that

$$s - g(s) = 0.$$

But we also want

$$f(s) = 0.$$

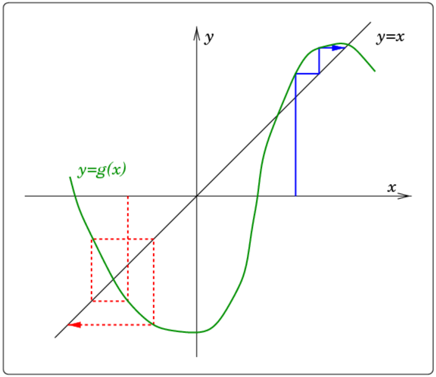
Therefore, *one possible* definition of the iterative map g is

$$g(x) = x - f(x).$$

Functional iteration or *fixed point* methods use maps of this kind.

Functional iteration graphically

- Roots of f are equivalent to points where $y = g(x)$ and $y = x$ intersect.
- The iteration process is given by the vertical / horizontal lines: given x_n, the next guess (x_{n+1}) is computed using g.
- Just because a fixed point exists there is no guarantee that the iteration process will converge to it!



- At a solution x^* of the equation we have $x^* = g(x^*)$. Assuming that g differentiable then for $x^{(n)} = x^* + e^{(n)}$ with small error $e^{(n)}$

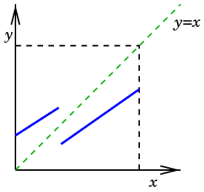
$$\begin{aligned} x^{(n+1)} &= g(x^{(n)}) = g(x^* + e^{(n)}) \approx g(x^*) + e^{(n)} g'(x^*) \\ \Longrightarrow e^{(n+1)} &\approx g'(x^*) e^{(n)} \end{aligned}$$

Hence iteration does not converge unless $|g'(x^*)| < 1$.

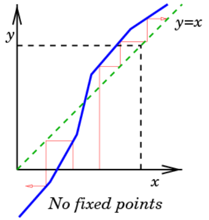
Geometric interpretations

- If g does not intersect $y = x$, no fixed points (top left).
- If range outside domain, sequence may not have a limit (top right).
- If slope less than 1, function has a unique fixed point (bottom right).

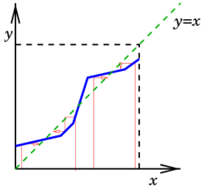
Not all fixed points *stable* (top right, bottom left): no numerical implementation would converge.



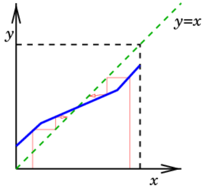
No fixed points



No fixed points



Two fixed points



Only one fixed point

Graphical examples - 1

Let us try to find the root of

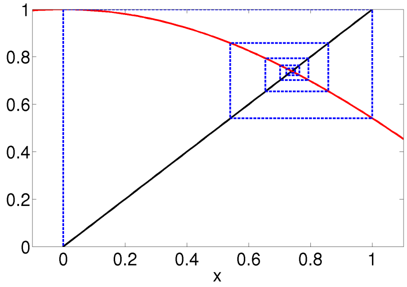
$$f(x) = x - \cos(x)$$

in the interval $[0, 1]$. The fixed point map is

$$\begin{aligned} g(x) &= x - f(x) \\ &= \cos(x), \end{aligned}$$

which obeys

1. $g(I) \subseteq I$, and
2. $|g'(x)| < 1$ within the interval.



The results of the iteration are

$$\begin{aligned} x_0 &= 0 & x_1 &= 1 \\ x_2 &= 0.540302 & x_{10} &= 0.731404 \\ x_{50} &= 0.739085 & x_{100} &= 0.739085. \end{aligned}$$

Graphical examples - 2

Let us try to find the root of

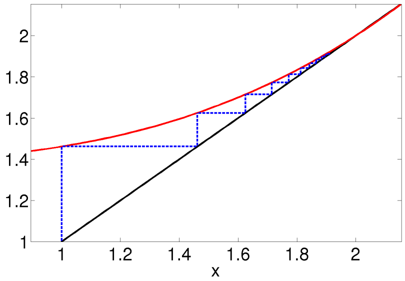
$$f(x) = x^3 - 13x + 18$$

in the interval $[1, 2.05]$. A map that works is

$$g(x) = \frac{x^3 + 18}{13}$$

which equals x at roots of f (check!) and obeys

1. $g(I) \subseteq I$, and
2. $|g'(x)| < 1$ within the interval.



The results of the iteration are

$$\begin{aligned} x_0 &= 1 & x_1 &= 1.461538 \\ x_2 &= 1.624768 & x_{10} &= 1.911737 \\ x_{50} &= 1.997695 & x_{100} &= 1.999958. \end{aligned}$$

From intuition to theory

Geometric intuition gives: a continuous iterative map, with the range within the domain, and slope less than 1, will have a unique fixed point. To prove this need:

Definition: A *contracting map* is a continuous map $g(x) : [a,b] = I \subseteq \mathbb{R} \to \mathbb{R}$ for which

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \ \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x) - g(y)| \leq L|x - y| \ \forall x, y \in I.$$

Note 1: if $g(x)$ is continuous and differntiable in $[a,b]$ and

$$\left|\frac{dg}{dx}\right| < L \quad \text{for} \ \ x \in [a,b],$$

then it follows that $g(x)$ is *Lipschitz* continuous with constant L.
Proof: by the Mean Value Theorem.

Note 2: if $g(x)$ is differentiable and *Lipschitz* continuous with constant L in $[a, b]$ then

$$\left|\frac{dg}{dx}\right| \leq L \quad \text{for} \quad x \in [a, b].$$

Existence

First theorem shows that a fixed point of a contracting map exists:

Theorem 1: If the function $g(x)$ is continuous in $I = [a, b]$ and $g(I) \subseteq I$, then $g(x)$ has at least one fixed point in I.

Theorem is stronger than we need; no need of Lipschitz continuity.

Proof:

- Construct continuous function $F(x) = g(x) - x$.
- Then since $a \leq g(x) \leq b$ it follows that $F(a) \geq 0$ and $F(b) \leq 0$.
- The intermediate value theorem then implies there is a point $s \in [a, b]$ where F vanishes.
- At this point $g(s) = s$ and there is a fixed point.

Uniqueness

Contraction mapping theorem: If $g(x)$ is a contraction mapping in I then there exists one and only one fixed point in I.

Proof where $g(x)$ also differentiable:

- Condition on Lifshitz continuity for contraction mapping implies

$$\left|\frac{dg}{dx}\right| \leq 1$$

- Hence $y = g(x)$ can cross $y = x$ at most one time.
- So solution of $x = g(x)$ for $x \in [a, b]$ is unique.

Speed of convergence

A fixed point may exist, but how fast will the iterative method find it?

Theorem: If $g(x)$ is a contracting map in I then, for arbitrary $x_0 \in I$ the sequence $x_{n+1} = g(x_n)$ converges to the unique fixed point s and the error $e_n = x_n - s$ obeys

$$|e_n| \leq \frac{L^n}{1-L}|x_1 - x_0|.$$

Theorem shows convergence depends strongly on L, hence magnitude of derivative. Smaller derivative gives faster convergence.

Speed of convergence

Proof :

- $x = s$ the solution satisfies $s = g(s)$.
- Lifschitz continuity condition then implies

$$\begin{aligned} |x_n - s| &= |g(x_{n-1}) - g(s)| \leq L|x_{n-1} - s| \\ |x_n - s| &\leq L^n|x_0 - s| \end{aligned} \tag{1}$$

- But

$$\begin{aligned} |x_0 - s| = |(x_0 - x_1) + (x_1 - s)| &\leq |x_1 - x_0| + |x_1 - s| \\ &\leq |x_1 - x_0| + L|x_0 - s| \\ \implies |x_0 - s| &\leq \frac{1}{1-L}|x_1 - x_0| \end{aligned}$$

- Substituting in (1)

$$|x_n - s| \leq \frac{L^n}{1-L}|x_1 - x_0|$$

Examples revisited

We looked at

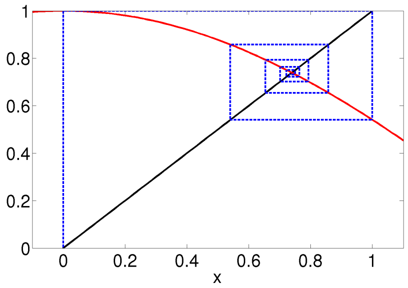
$$g(x) = \cos(x)$$

in the interval $[0, 1]$. It is a contracting map, but

$$\max_{x \in I} |g'(x)| = \sin(1)$$

$$= 0.84147.$$

We should not expect fast convergence.



The results of the iteration are

$$x_0 = 0 \qquad x_1 = 1$$
$$x_2 = 0.540302 \qquad x_{10} = 0.731404$$
$$x_{50} = 0.739085 \qquad x_{100} = 0.739085.$$

Examples revisited

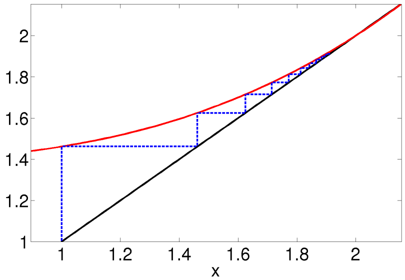
We looked at

$$g(x) = \frac{x^3 + 18}{13}$$

in the interval $[1, 2.05]$. The map is monotonic and the largest value of the derivative is

$$|g'(2.05)| = 0.97!$$

We therefore expect very slow convergence.



The results of the iteration are

$$\begin{aligned} x_0 &= 1 & x_1 &= 1.461538 \\ x_2 &= 1.624768 & x_{10} &= 1.911737 \\ x_{50} &= 1.997695 & x_{100} &= 1.999958. \end{aligned}$$

Summary

- Fixed point or functional iteration methods for finding the root of $f(x)$ look for fixed points of

$$g(x) = x - f(x).$$

- Typically, the faster the scheme converges the closer the initial guess needs to be.
- Existence and uniqueness of the root found can be proved if g is a contraction mapping for which
  1. $g(I) \subseteq I$
  2. g is Lipschitz continuous with $L < 1$.
- The speed of convergence is related to L - the smaller L, the faster the convergence.
- **All the results refer to the map g, not the original function f. The function g can be chosen freely, provided the fixed point is a root of f!**

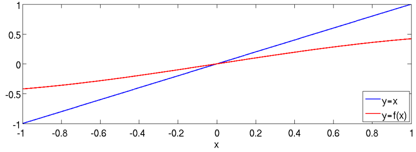
Does it contract? Question 1

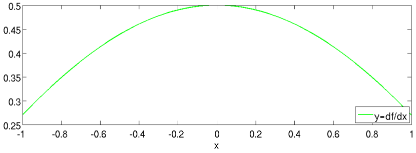
Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \;\; \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x) - g(y)| \leq L|x - y| \;\; \forall x, y$$

Does it contract?





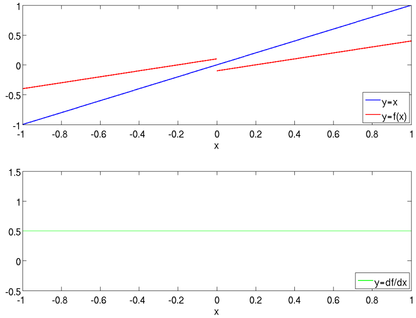
Does it contract? Question 2

Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \;\; \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x)-g(y)| \leq L|x-y| \;\; \forall x, y$$

Does it contract?



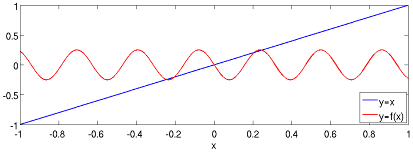
Does it contract? Question 3

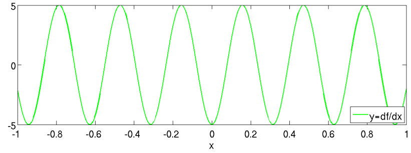
Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \ \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x) - g(y)| \leq L|x - y| \ \forall x, y$$

Does it contract?





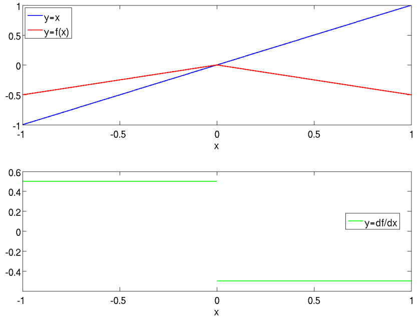
Does it contract? Question 4

Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \ \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x)-g(y)| \leq L|x-y| \ \forall x, y$$

Does it contract?



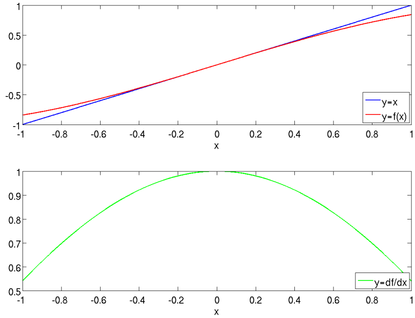
Does it contract? Question 5

Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \ \ \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x) - g(y)| \leq L|x - y| \ \ \forall x, y$$

Does it contract?



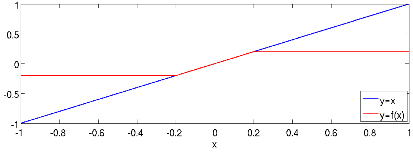
Does it contract? Question 6

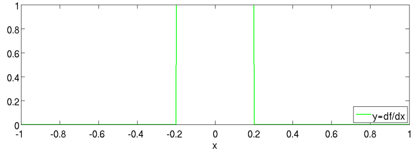
Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \ \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x) - g(y)| \leq L|x - y| \ \forall x, y$$

Does it contract?



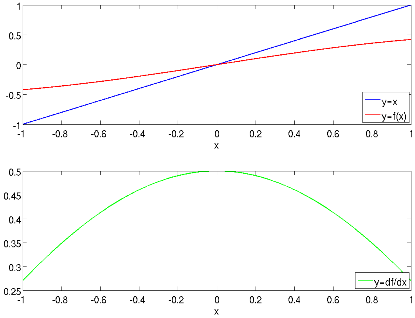


Does it contract? Answer 1

Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \ \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x) - g(y)| \leq L|x - y| \ \ \forall x, y$$



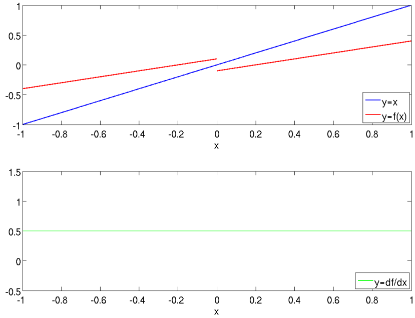
Does it contract? Yes! Both the conditions are met.

Does it contract? Answer 2

Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \;\; \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x) - g(y)| \leq L|x - y| \;\; \forall x, y$$



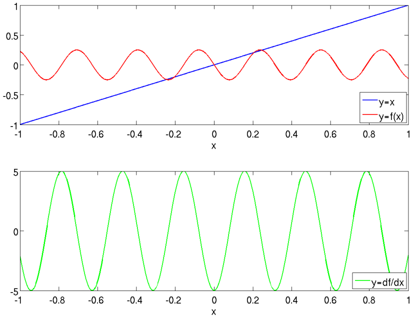
Does it contract? No! There are no fixed points of the map. The continuity condition has failed.

Does it contract? Answer 3

Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \;\; \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x) - g(y)| \leq L|x - y| \;\; \forall x, y$$



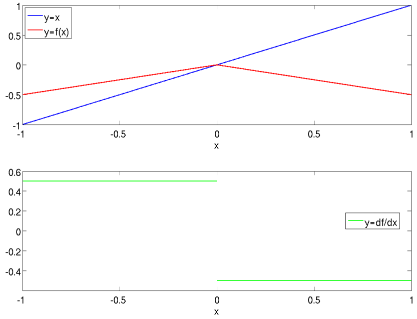
Does it contract? No! Although the function is contained in its interval the second condition is not met.

Does it contract? Answer 4

Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \;\; \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x)-g(y)| \leq L|x-y| \;\; \forall x, y$$



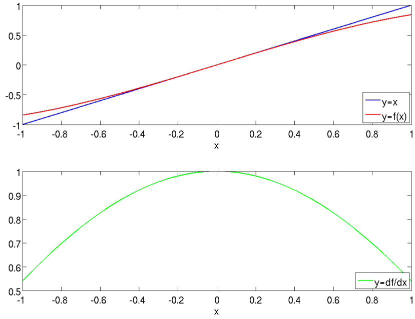
Does it contract? Yes! Even though there is a jump in the derivative both conditions are met.

Does it contract? Answer 5

Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \;\; \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x) - g(y)| \leq L|x - y| \;\; \forall x, y$$



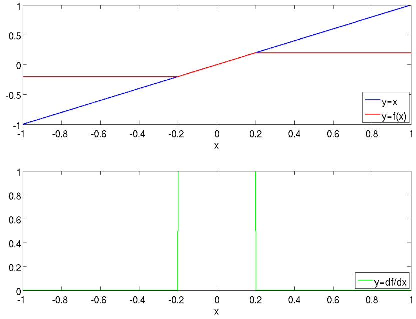
Does it contract? No! The derivative is 1 and we cannot guarantee that there is only one fixed point.

Does it contract? Answer 6

Definition: A continuous map is a *contracting map* for $g(x) : [a, b] = I \subseteq \mathbb{R} \to \mathbb{R}$ if

1. $g(I) \subseteq I \Leftrightarrow g(x) \in I \;\; \forall x \in I$;
2. $g(x)$ is *Lipschitz* continuous with constant $L < 1$:

$$|g(x) - g(y)| \leq L|x - y| \;\; \forall x, y$$



Does it contract? No! The derivative is 1 and we cannot guarantee that there is only one fixed point. In this case there is a whole line of fixed points.

Chapter 7 - Example Iteration Methods and Speed

G. Richardson

MATH3018/6141, Semester 1

Brief reminders

Previously: finding root of f, i.e. solution to

$$f(x) = 0.$$

Iteration: define sequence $\{x_n\}$ using map $g : x_n \to x_{n+1}$. Map designed such that fixed points of g are roots of f. If sequence converges, have found a root.

For a contraction mapping with Lipschitz constant $L < 1$ map converges, with error

$$|e_n| \leq \frac{L^n}{1-L}|x_1 - x_0|.$$

Error depends on worst case value of L in whole interval. Often find faster convergence for better initial guesses.

Speed of convergence

Expect speed of convergence to depend on map g near root s. Can show this for "nice" g and "good" x_n. Define error at n^{th} stage

$$e_n = x_n - s.$$

As s is a fixed point of g, use Taylor expansion of e_{n+1}:

$$\begin{aligned} e_{n+1} = x_{n+1} - s &= g(x_n) - g(s) \\ &= g'(s)(x_n - s) + \tfrac{1}{2!}g''(s)(x_n - s)^2 + \dots \\ &= g'(s)e_n + \tfrac{1}{2!}g''(s)e_n^2 + \dots \end{aligned}$$

If error small then a *linear* or *first order* method converges as

$$e_{n+1} \sim g'(s)e_n;$$

decreases by a constant amount each step, proportional to the derivative of the map at the root.

Speed of convergence

Expect speed of convergence to depend on map g near root s. Can show this for "nice" g and "good" x_n. Define error at n^{th} stage

$$e_n = x_n - s.$$

As s is a fixed point of g, use Taylor expansion of e_{n+1}:

$$\begin{aligned} e_{n+1} = x_{n+1} - s &= g(x_n) - g(s) \\ &= g'(s)(x_n - s) + \tfrac{1}{2!}g''(s)(x_n - s)^2 + \dots \\ &= g'(s)e_n + \tfrac{1}{2!}g''(s)e_n^2 + \dots \end{aligned}$$

If $g'(s) \equiv 0$ and error small a *quadratic* or *second order* map converges as

$$e_{n+1} \sim g''(s)e_n^2.$$

The more derivatives of g that vanish at the root, the faster the convergence (but the better the guess must be).

General framework

As usual we are trying to solve

$$f(x) = 0$$

for $x \in [a, b]$. Functional iteration is map

$$g(x) = x - f(x);$$

at fixed points of this map $f(x)$ must vanish.

Simple generalization: introduce $\varphi(x) \neq 0$, then

$$g(x) = x - \varphi(x)f(x)$$

also has that $g(s) = s \implies f(s) = 0 \implies$ root.

Increase speed of convergence by choice of φ, provided

$$0 < \varphi(x) < \infty, \quad x \in [a, b].$$

Chord method

Simplest choice: $\varphi(x) = m \neq 0$, giving

$$g(x) = x - mf(x).$$

This is known as the *chord method*.
Free to choose m, provided that g is a contraction mapping in $[a, b]$.
Requires $|g'(x)| < 1$ in interval; implies

$$\begin{aligned} & |1 - mf'(x)| < 1 && \forall x \in [a, b] \\ \Rightarrow \quad & 0 < mf'(x) < 2 && \forall x \in [a, b]. \end{aligned}$$

So if $m = 1/f'(s)$ then method converges quadratically.

Choose m such that $0 < mf'(x) < 2 \ \forall x \in [a, b]$ and it provides a good guess for $1/f'(s)$. Iterate with this fixed value of m.

Chord method: geometric picture

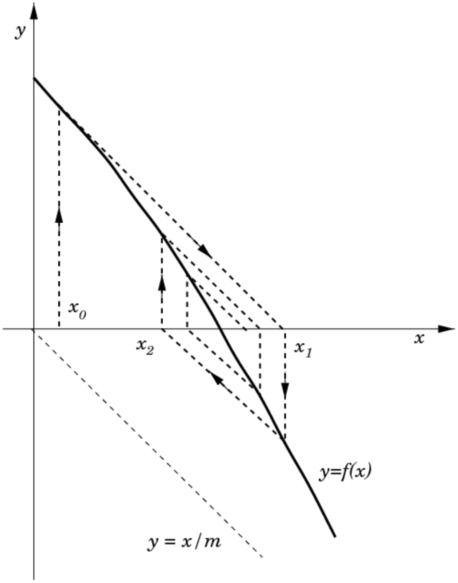
The chord method

$$x_{n+1} = x_n - mf(x_n).$$

Here we use a different geometric picture to previously.
Consider the line between the points $(x, y) = (x_n, f(x_n))$ and $(x, y) = (x_{n+1}, 0)$

$$y = \frac{x - x_{n+1}}{x_n - x_{n+1}} f(x_n) = \frac{x - x_{n+1}}{m}.$$

This line is a chord with gradient $1/m$ from the curve $y = f(x)$ at $x = x_n$ which intersects the x-axis at the new point $x = x_{n+1}$.
Plot uses $f(x)$ NOT $g(x)$!



Example of the chord method

We look at

$$f(x) = x - \cos(x), \quad x \in [0, 1].$$

The generic chord method has map

$$g(x) = x(1 - m) + m\cos(x),$$

and the range of allowable m values is

$$0 < |m + m\sin(x)| < 2$$

implying $0 < m < 2$ and $0 < 1.8415m < 2$. Choosing $m = 1.08\,(\lesssim 2/1.8415)$ we obtain the iteration map

$$x_{n+1} = -0.08x_n + 1.08\cos(x_n).$$

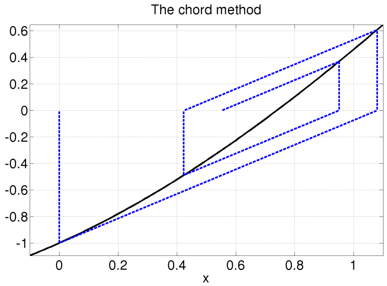
Example of the chord method - 2

Sequence produced by map

$$x_{n+1} = -0.08x_n + 1.08\cos(x_n)$$

starting from zero is

$$\begin{aligned} x_0 &= 0 \\ x_1 &= 1.0800 \\ x_2 &= 0.4226 \\ x_3 &= 0.9512 \\ x_4 &= 0.5511 \\ x_5 &= 0.8760 \end{aligned}$$



Example of the chord method - 2

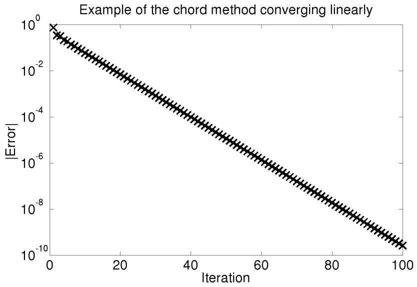
Sequence produced by map

$$x_{n+1} = -0.08x_n + 1.08\cos(x_n)$$

starting from zero is

$$\begin{aligned} x_0 &= 0 \\ x_1 &= 1.0800 \\ x_2 &= 0.4226 \\ x_3 &= 0.9512 \\ x_4 &= 0.5511 \\ x_5 &= 0.8760 \end{aligned}$$

converges linearly to solution $s = 0.739085\ldots$.



Newton's method

For quadratic convergence need $g'(s) \equiv 0$. Newton's method: choose

$$\varphi(x) = \frac{1}{f'(x)} \quad \Longrightarrow \quad g(x) = x - \frac{f(x)}{f'(x)}.$$

A simple check shows

$$g'(s) = 1 - 1 + \frac{f(s)f''(s)}{(f'(s))^2} = 0.$$

The iteration scheme is

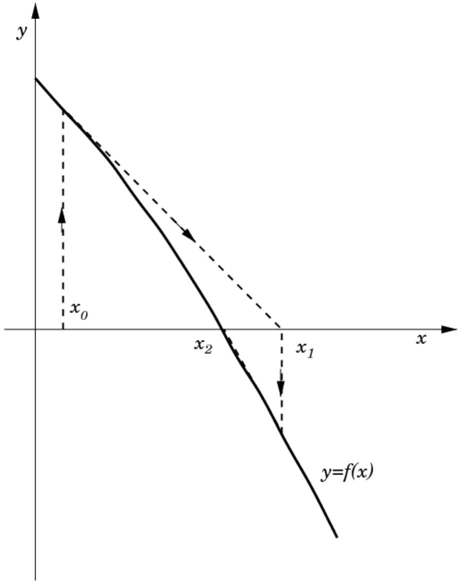
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)};$$

more complex, has limitations:

- Needs derivative. If unknown, better to use secant method.
- If derivative vanishes at root, speed of convergence calculation is wrong. Problem for multiple roots, for which Newton's method converges at best linearly.

Newton's method: geometric picture

Geometric picture for Newton's method: move on lines with slope given by derivative at x_n.



Example of Newton's method

We again look at

$$f(x) = x - \cos(x), \quad x \in [0, 1].$$

Newton's method has map

$$\begin{aligned} g(x) &= x - \frac{f(x)}{f'(x)} \\ &= \frac{x \sin(x) + \cos(x)}{1 + \sin(x)}. \end{aligned}$$

As $f' \neq 0$ in the domain, expect quadratic convergence.

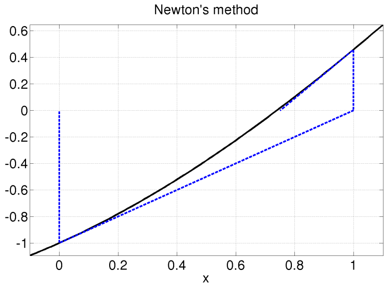
Example of Newton's method - 2

Sequence produced by map

$$x_{n+1} = \frac{x_n \sin(x_n) + \cos(x_n)}{1 + \sin(x_n)},$$

starting from zero is

$$\begin{aligned} x_0 &= 0.0000000000 \\ x_1 &= 1.0000000000 \\ x_2 &= 0.7503638678 \\ x_3 &= 0.7391128909 \\ x_4 &= 0.7390851334 \\ x_5 &= 0.7390851332 \end{aligned}$$



Example of Newton's method - 2

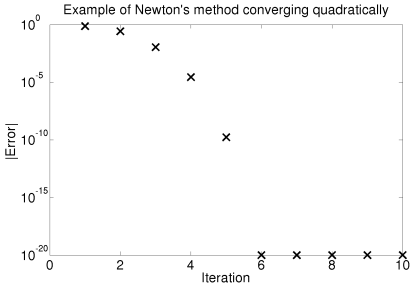
Sequence produced by map

$$x_{n+1} = \frac{x_n \sin(x_n) + \cos(x_n)}{1 + \sin(x_n)},$$

starting from zero is

$$\begin{aligned} x_0 &= 0.0000000000 \\ x_1 &= 1.0000000000 \\ x_2 &= 0.7503638678 \\ x_3 &= 0.7391128909 \\ x_4 &= 0.7390851334 \\ x_5 &= 0.7390851332 \end{aligned}$$

converges quadratically to solution $s = 0.739085\ldots$.



Secant method

Key disadvantage of Newton's method: needs derivative $f'(x)$. Extra function evaluations add cost, even where derivative can be computed. The Secant Method instead approximates derivative using

$$f'(x_n) \simeq \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

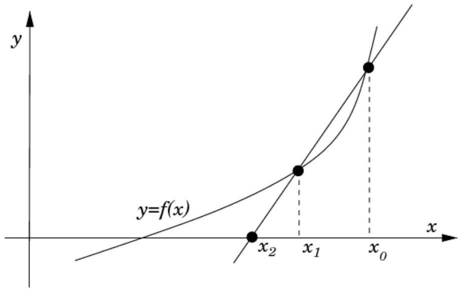
This results in the iteration scheme

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

Does not match framework: *two* previous values in sequence. Means two initial guesses required, and contraction mapping theorems cannot be used. Can show that method converges; often more useful and faster than Newton's method.

Secant method: geometric picture

Geometric picture for Secant method: move on lines whose slope is determined by secant to function through x_{n-1}, x_n.



Example of the Secant method

Again look at

$$f(x) = x - \cos(x), \quad x \in [0, 1].$$

The Secant method has the iteration scheme (no map!)

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

Expect slightly slower convergence than Newton's method.

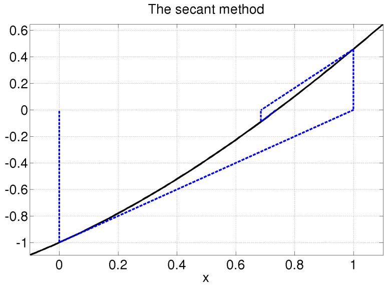
Example of the Secant method - 2

Sequence uses iteration

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

starting from $x_0 = 0, x_1 = 1$:

$$x_0 = 0.0000000000$$
$$x_1 = 1.0000000000$$
$$x_2 = 0.6850733573$$
$$x_3 = 0.7362989976$$
$$x_4 = 0.7391193619$$
$$x_5 = 0.7390851121$$



Example of the Secant method - 2

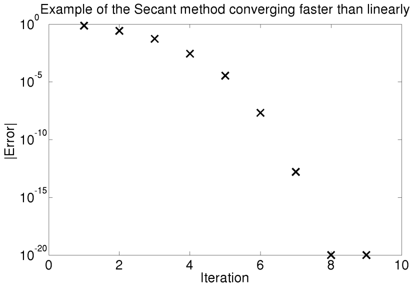
Sequence uses iteration

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

starting from $x_0 = 0, x_1 = 1$:

$$x_0 = 0.0000000000$$
$$x_1 = 1.0000000000$$
$$x_2 = 0.6850733573$$
$$x_3 = 0.7362989976$$
$$x_4 = 0.7391193619$$
$$x_5 = 0.7390851121$$

converges to solution
$s = 0.739085\ldots$.



Summary

- The speed of convergence from the error bound using the Lipschitz constant L is the worst-case scenario. Normally we expect the speed of convergence to be determined by properties of the map near the root s.
- By generalizing our iterative map to

  $$g(x) = x - \varphi(x)f(x), \quad \varphi(x) \neq 0$$

  we still have that fixed points of g are roots of f, but can speed convergence by varying φ.
- The more derivatives of g that vanish at the root, the faster the convergence, but the better the initial guess must be.
- The chord method and Newton's method fit in this general framework and converge linearly and quadratically respectively.
- The secant method does not fit in this framework. It takes more iterations to converge than Newton's method, but is often faster in practice.

Chapter 8- Roots of Nonlinear Systems

G. Richardson

MATH3018/6141, Semester 1

Systems of nonlinear equations

A considerably more difficult problem than solving the single equation

$$f(x) = 0$$

for the root s of the single variable x, is solving the system size n:

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}.$$

In general case may not be possible to check existence of root, let alone uniqueness.

Where possible we shall extend the previous results.

Simple example

Consider the system

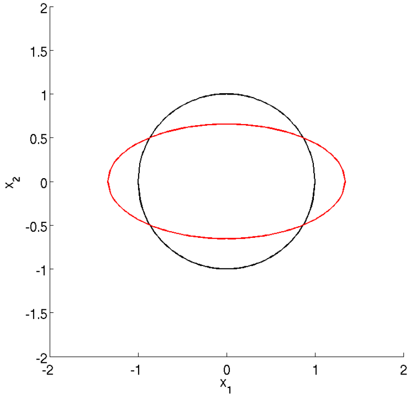
$$x_1^2 + x_2^2 - 1 = 0$$
$$5x_1^2 + 21x_2^2 - 9 = 0.$$

Have $\boldsymbol{x} = (x_1, x_2)^T$, $\mathbf{f} = (f_1, f_2)^T$

$$f_1(x_1, x_2) = x_1^2 + x_2^2 - 1,$$
$$f_2(x_1, x_2) = 5x_1^2 + 21x_2^2 - 9.$$

Four solutions $\xi = (\pm\sqrt{3}/2, \pm 1/2)^T$. Match the four intersections of the two curves.

Functional iteration $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x}$ fails!



Contraction mapping

Again find root $\boldsymbol{s}$ by constructing sequence $\{\boldsymbol{x}_n\}$ using map $\mathbf{g} : \mathbb{R}^N \to \mathbb{R}^N$:

$$\boldsymbol{x}_{n+1} = \boldsymbol{g}(\boldsymbol{x}_n)$$

In analogy with scalar case define interval I as

$$I = \{\boldsymbol{x} \in \mathbb{R}^N \mid a_j < x_j < b_j,\ j = 1, 2, \ldots, n\}.$$

Hence define contraction map as a Lipschitz continuous map $\boldsymbol{g}(\boldsymbol{x})$ such that $\boldsymbol{g}(I) \subseteq I$ and with Lipschitz constant $L < 1$:

$$\|\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\| \quad \forall \boldsymbol{x}, \boldsymbol{y} \in I.$$

Remark: If the mapping $\boldsymbol{g}(\boldsymbol{x})$ is differentiable then it is Lipschitz continuous with Lipschitz constant L provided the Jacobian matrix J, with coefficients

$$J_{ij} = \frac{\partial g_i}{\partial x_j}$$

satisfies the condition $\|J\| \leq L$ for any matrix norm.

Contraction mapping theorems

Theorem: If $\boldsymbol{g}$ is continuous in I and $\boldsymbol{g}(I) \subseteq I$ then $\boldsymbol{g}$ has at least one fixed point in I.

Contraction mapping theorem in $\mathbb{R}^N$: If $\boldsymbol{g}(\boldsymbol{x})$ is a contraction mapping in I then there exists one and only one fixed point in I.

Theorem: If $\boldsymbol{g}(\boldsymbol{x})$ is a contraction mapping in I then for arbitrary $\boldsymbol{x}_0 \in I$ then the sequence $\{\boldsymbol{x}_n\}$ converges to the unique fixed point with error

$$\|\boldsymbol{e}_n\|_\infty \leq \frac{L^n}{1-L}\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|_\infty .$$

Remark: If $\boldsymbol{g}(\boldsymbol{x})$ is differentiable in I and

$$\left|\frac{\partial g_i}{\partial x_j}\right| \leq \frac{L}{N}, \quad \text{for } \boldsymbol{x} \in I \quad \text{and } L < 1$$

then $\|J\|_1 \leq L$ and $\|J\|_\infty \leq L$ so that it is a contraction map and there is one and only one fixed point in I.

Example

Look at map

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{cases} g_1(x_1, x_2, x_3) = \frac{1}{3}\cos(x_2 x_3) + \frac{1}{6} \\ g_2(x_1, x_2, x_3) = \frac{1}{9}\sqrt{x_1^2 + \sin x_3 + 1.06} - 0.1 \\ g_3(x_1, x_2, x_3) = -\frac{1}{20}\exp(-x_1 x_2) - (10\pi - 3)/60 \end{cases}.$$

By computing the Jacobian of the map

$$J(\boldsymbol{x}) = \begin{pmatrix} 0 & -\frac{1}{3}\sin(x_2 x_3) x_3 & -\frac{1}{3}\sin(x_2 x_3) x_2 \\ \frac{1}{9}\frac{x_1}{\sqrt{x_1^2+\sin(x_3)+1.06}} & 0 & \frac{1}{18}\frac{\cos(x_3)}{\sqrt{x_1^2+\sin(x_3)+1.06}} \\ \frac{1}{20} x_2 \exp(-x_1 x_2) & \frac{1}{20} x_1 \exp(-x_1 x_2) & 0 \end{pmatrix}$$

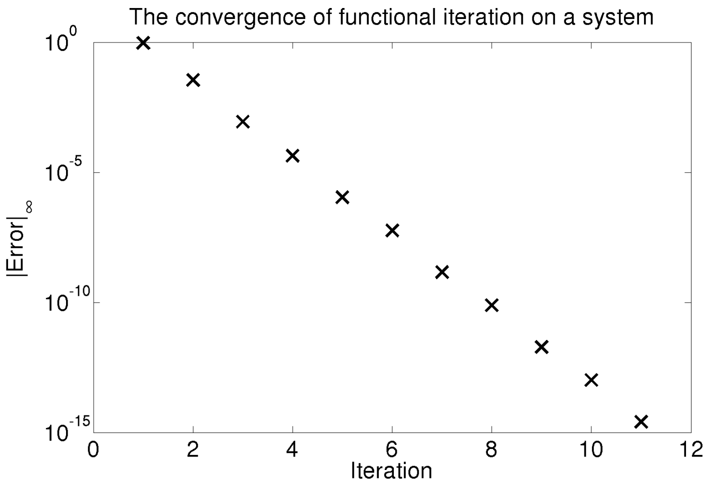
see that all elements of J are less than $1/3$ in magnitude in $I = [-1, 1]^3$. Hence there is a unique fixed point in I.

Example: 2

Iterate starting from $\boldsymbol{x} = (0.2, 0.3, 0.4)$: appears to converge to a fixed point

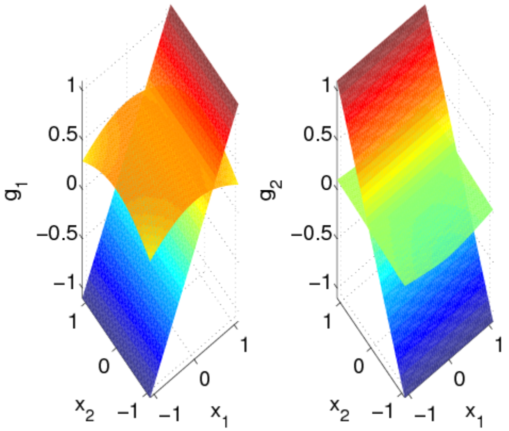
n	x_1	x_2	x_3
0	0.2	0.3	0.4
1	0.4976029	0.0356019	-0.5206870
2	0.4999427	0.0000082	-0.5227208
3	0.5000000	0.0000434	-0.5235986
4	0.5000000	0.0000000	-0.5235977
5	0.5000000	0.0000001	-0.5235988

Example: 3



Convergence is linear, as usual. Geometric picture not straightforward.

Geometric picture



Trying to find unique point giving the intersection of *each* map with the appropriate axis plane; we move along each in turn.

(Non)linear systems analogies

Consider e.g. Jacobi for $A\boldsymbol{x} = \boldsymbol{b}$: constructs sequence $\{\boldsymbol{x}_n\}$ with correct answer in the limit. Analogous to iterative methods here. Inspires:

1. "Gauss-Seidel": use guess as soon as possible. I.e.,

$$\begin{aligned} & (x_1)_{n+1} = g_1\left((x_1)_n, (x_2)_n\right) \\ \text{then} \qquad & (x_2)_{n+1} = g_2\left((x_1)_{n+1}, (x_2)_n\right). \end{aligned}$$

2. "Relaxation": $\hat{\boldsymbol{x}}_{n+1} = \boldsymbol{g}(\boldsymbol{x}_n)$, but *actual* next iterate uses "correction" as

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \omega(\hat{\boldsymbol{x}}_{n+1} - \boldsymbol{x}_n).$$

   Typically take $\omega < 1$ to promote convergence (*under* relaxation); can be impractical (slow).

Newton's method – graphical approach

"Best" method in scalar case: Newton iteration.

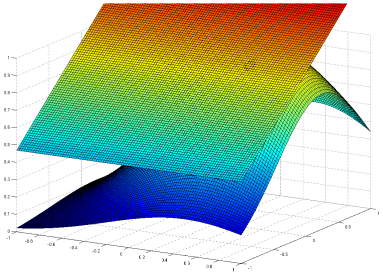
Scalar geometric picture: follow tangent from curve to axis. Derive using Taylor expansion. Where $x = s$ is solution $f(s) = 0$. But

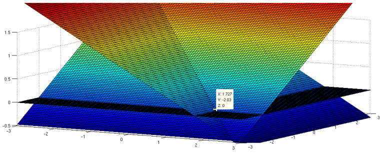
$$f(s) = f(x_n) + (s - x_n)f'(x_n) + \cdots$$
$$\Longrightarrow s \approx x_n - \left(f'(x_n)\right)^{-1} f(x_n).$$

Suggests iterative scheme

$$x_{n+1} = x_n - \left(f'(x_n)\right)^{-1} f(x_n)$$

Geometric picture: follow tangent *planes*. Apply to each component (map). Find mutual intersections.





Newton's method from Taylor series

Solution $\boldsymbol{x} = \boldsymbol{s}$ to 2-d problem $\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$. Expand about $\boldsymbol{x} = \boldsymbol{x}_n$

$$f_1(s_1, s_2) = 0 = f_1(x_{1,n}, x_{2,n}) + (s_1 - x_{1,n})\frac{\partial f_1}{\partial x_1} + (s_2 - x_{2,n})\frac{\partial f_1}{\partial x_2} + \cdots,$$
$$f_2(s_1, s_2) = 0 = f_2(x_{1,n}, x_{2,n}) + (s_1 - x_{1,n})\frac{\partial f_2}{\partial x_1} + (s_2 - x_{2,n})\frac{\partial f_2}{\partial x_2} + \cdots, .$$

Assume $(x_{1,n}, x_{2,n})$ lies close to (s_1, s_2) then

$$\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix} \begin{pmatrix} s_1 - x_{1,n} \\ s_2 - x_{2,n} \end{pmatrix} \approx \begin{pmatrix} -f_1(x_{1,n}, x_{2,n}) \\ -f_2(x_{1,n}, x_{2,n}) \end{pmatrix}.$$

This can be written as a matrix equation

$$J(\boldsymbol{x}_n) \cdot (\boldsymbol{s} - \boldsymbol{x}_n) \approx -(\boldsymbol{f}(\boldsymbol{x}_n))$$

where J is the *Jacobian* matrix. And suggests the iteration scheme

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - (J^{-1}(\boldsymbol{x}_n))\boldsymbol{f}(\boldsymbol{x}_n).$$

Newton's method algorithm

Final algorithm. Given guess $\boldsymbol{x}_0 \implies \boldsymbol{x}_n$. At step $n+1$:

1. Compute $\boldsymbol{f}(\boldsymbol{x}_n)$.
2. Compute Jacobian $J(\boldsymbol{x}_n)$.
3. Solve for "correction" $\boldsymbol{c}$ from $J(\boldsymbol{x}_n)\boldsymbol{c} = -\boldsymbol{f}(\boldsymbol{x}_n)$.
4. Compute $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{c}$.

Never invert Jacobian explicitly.

Newton's method is computationally expensive: generalization of secant method exists.

Simple example revisited

Consider the system

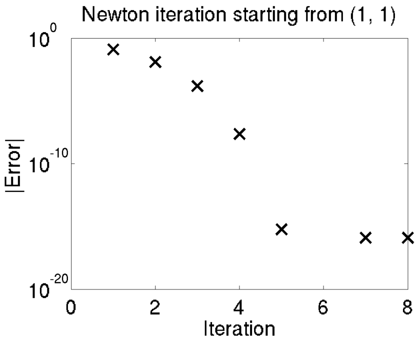
$$x_1^2 + x_2^2 - 1 = 0$$
$$5x_1^2 + 21x_2^2 - 9 = 0.$$

Have $\boldsymbol{x} = (x_1, x_2)^T$, $\mathbf{f} = (f_1, f_2)^T$

$$f_1(x_1, x_2) = x_1^2 + x_2^2 - 1,$$
$$f_2(x_1, x_2) = 5x_1^2 + 21x_2^2 - 9.$$

Four solutions $\xi = (\pm\sqrt{3}/2, \pm 1/2)^T$. Match the four intersections of the two curves.

Newton's method from $(1, 1)$ works.



Summary

- The theory of contraction mapping essentially carries over to systems by replacing scalars with vectors and absolute values with norms.
- Analogies with solving systems of *linear* equations leads to “Gauss-Seidel” and “relaxation” methods.
- Newton’s method generalizes by considering tangent *(hyper)planes*, which leads to a method based on the Jacobian matrix.

Chapter 9 - Quadrature

G. Richardson

MATH3018/6141, Semester 1

Numerical Quadrature

The aim of numerical quadrature is to compute

$$\int_a^b f(x)\,\mathrm{d}x$$

where f is a real function of a single variable x.

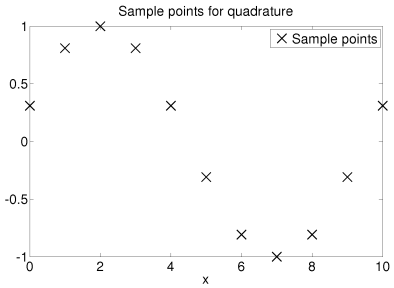
More complex integrals are best evaluated by analytical reduction to a (set of) integral(s) in the form above.

Here we will always assume that $f(x)$ is finite within $[a, b]$, and frequently we will implicitly assume that it is differentiable.

The simplest quadrature

Standard situation: the value of the function is known (or can be evaluated) at a set of points or *nodes*.

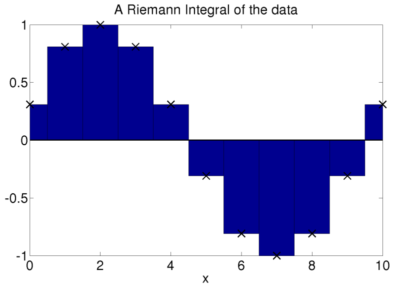
The simplest evaluation of the quadrature uses the Riemann integral: a rectangular area is computed for each node, and the total integral is the sum of these areas.



The simplest quadrature

Standard situation: the value of the function is known (or can be evaluated) at a set of points or *nodes*.

The simplest evaluation of the quadrature uses the Riemann integral: a rectangular area is computed for each node, and the total integral is the sum of these areas.

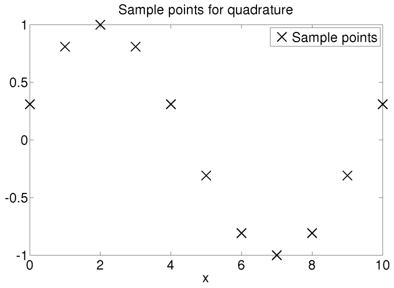


Function representation

- The simple rule from the Riemann integral is not very accurate.
- Most improvements change how the function is represented using a finite amount of information.

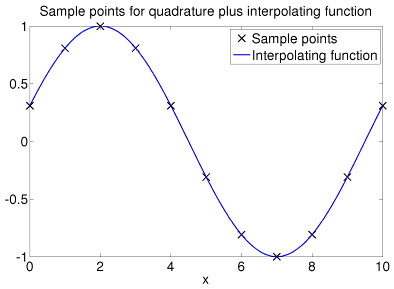
Simplest assumption:

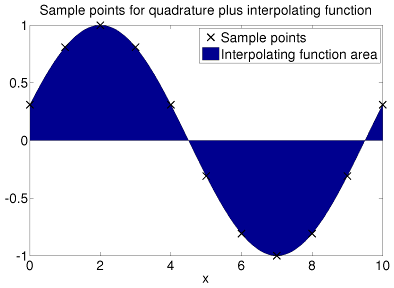
- Know the *values* $f(x_j)$ at nodes x_j.
- This does not give complete information about the function.



Function representation

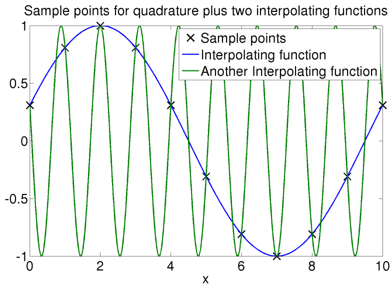
- Know function $f(x)$ at nodes x_j.
- Does not give complete information about the function.
- Need to approximate the function away from the nodes.
- *Assume* the function behaves in a simple way and can be approximated by a simple *interpolating function*.
- If the interpolating function is integrable then we are done; compute the integrals between each node and sum.





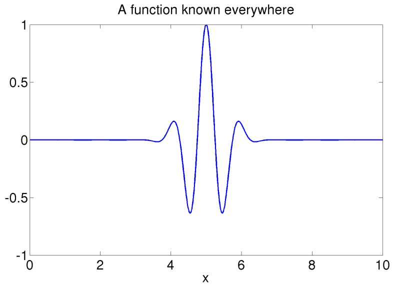
Function representation

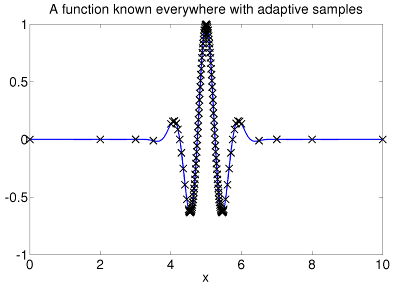
The choice of interpolating function is not unique. With quadrature this is not usually much of an issue.



Function representation

- The function can be evaluated at any point in the interval.
- In order to save computation cost we may wish to do this as little as possible.
- By picking the nodes and the interpolating function appropriately can obtain optimal accuracy for a given number of nodes.
- Requires that nodes are chosen based on the behaviour of the function f.
- This is an *adaptive* algorithm.





Polynomial interpolation

The *Newton-Cotes* formulae are a simple and robust class of algortihms that make the following assumptions:

- The function $f(x)$ is known (or can be evaluated) at a finite set of nodes $\{x_j\}$, with $j = 0, \ldots, N$, in the interval $[a, b]$.
- An interpolating function $g(x)$ is found that passes through these points, i.e.

$$g(x_j) = f(x_j)\left(= f_j\right), \quad j = 0, \ldots, N.$$

- The interpolating function is a polynomial or piecewise polynomial.
- The integral is computed from the exact integral of $g(x)$ over the interval.

Different methods are based on the order of the polynomials used and on the restrictions of the nodes.

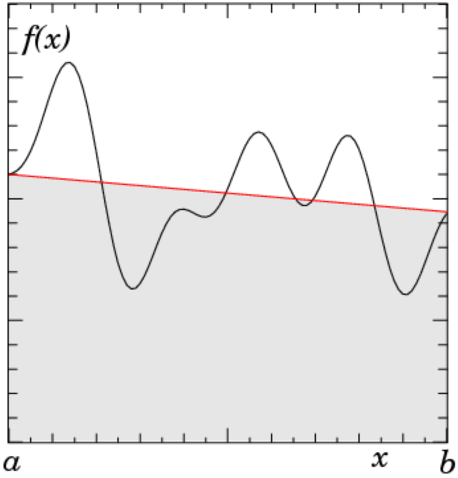
Trapezoidal rule

The trapezoidal rule uses an interpolating polynomial $g(x)$ of order 1: *i.e.* a straight line.

Hence

$$\int_a^b f(x)\,\mathrm{d}x = \tfrac{1}{2}(b-a)\,[f(b)+f(a)]\,.$$

Very inaccurate unless the interval is small or $f(x)$ is almost linear.



Trapezoidal rule

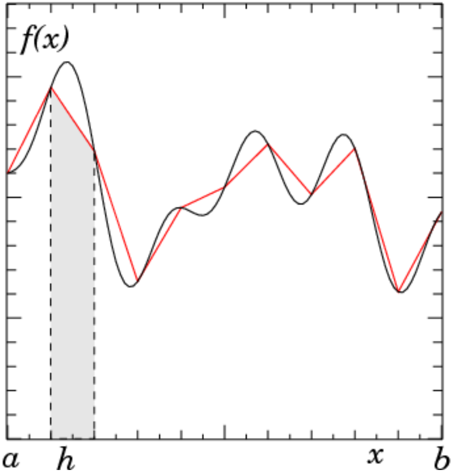
The trapezoidal rule uses an interpolating polynomial $g(x)$ of order 1: *i.e.* a straight line.

Accuracy obtained by using *composite* trapezoidal rule. Divide interval into N equal subintervals length

$$h = (b - a)/N,$$

and apply trapezoidal rule to each. Resulting interpolation polynomial g is *piecewise* linear. Area in each subinterval is

$$A_j = \frac{1}{2}(x_j - x_{j-1})\left[f(x_j) + f(x_{j-1})\right].$$



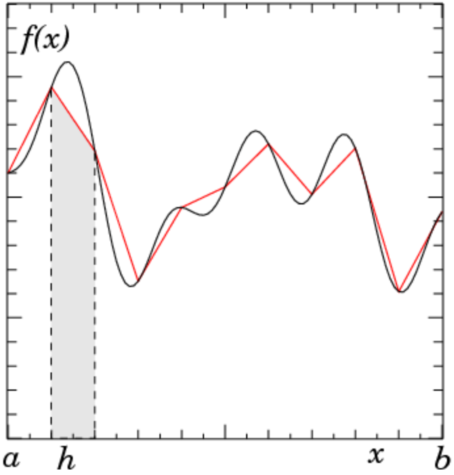
Composite Trapezoidal Rule

From each subinterval the area is

$$A_j = \tfrac{1}{2}(x_j - x_{j-1})\left[f(x_{j-1}) + f(x_j)\right]$$
$$= \tfrac{h}{2}\left[f_{j-1} + f_j\right].$$

Obtain full answer by summing

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{j=1}^{N} A_j$$
$$= \frac{h}{2}\sum_{j=1}^{N}\left[f_{j-1} + f_j\right]$$



Often written as

$$\int_a^b f(x)\,\mathrm{d}x = \tfrac{h}{2}\left(f_0 + 2f_1 + 2f_2 + \cdots + 2f_{N-1} + f_N\right).$$

Errors: 1

To compute the error from a composite quadrature rule we bound the error in any subinterval. This will depend on the width of the interval, h.

We firstly write the trapezoidal rule on one interval as

$$A_j = \frac{h}{2}\left(f_j + f_{j+1}\right) = \frac{h}{2}\left(f(x_j) + f(x_j + h)\right).$$

If we Taylor expand $f(x_{j+1})$ about x_j we have

$$A_j = hf(x_j) + \frac{h^2}{2}f'(x_j) + \frac{h^3}{4}f^{(2)}(x_j) + \dots$$

Errors: 2

To find the error, we need the exact result in powers of h. We define

$$F(t) = \int_{x_j}^{x_j+t} f(x)\,\mathrm{d}x.$$

The quantity

$$F(h) = \int_{x_j}^{x_j+h} f(x)\,\mathrm{d}x,$$

is what we try to compute.

We note that

$$\frac{\mathrm{d}^n F}{\mathrm{d}t^n} = f^{(n-1)}(x_j + t)$$

or in particular

$$\frac{\mathrm{d}F}{\mathrm{d}t} = f(x_j + t).$$

Errors: 3

We Taylor expand $F(t)$ about $t = 0$

$$F(h) = F(0) + h\,\frac{\mathrm{d}F}{\mathrm{d}t}\bigg|_{t=0} + \frac{h^2}{2}\,\frac{\mathrm{d}^2F}{\mathrm{d}t^2}\bigg|_{t=0} + \frac{h^3}{6}\,\frac{\mathrm{d}^3F}{\mathrm{d}t^3}\bigg|_{t=0} + \dots$$

From the definition of $\mathrm{d}F/\mathrm{d}t$ etc. we have

$$F(h) = hf(x_j) + \tfrac{h^2}{2}f'(x_j) + \tfrac{h^3}{6}f^{(2)}(x_j) + \dots.$$

Comparing with the trapezoidal rule

$$A_j = hf(x_j) + \tfrac{h^2}{2}f'(x_j) + \tfrac{h^3}{4}f^{(2)}(x_j) + \dots$$

the error in the subinterval is

$$|A_j - F(h)| \le \frac{h^3}{12}|f^{(2)}(x_j)|.$$

Summing over all N subintervals and using $hN = (b-a)$ gives

$$\text{Error} \le \frac{(b-a)h^2}{12}M_2, \quad M_2 = \max_{x\in[a,b]}|f^{(2)}(x)|.$$

Example

We look at

$$\int_0^{\pi/2} \sin(x)\, dx$$

using the trapezoidal rule. The exact answer is 1.
With three points $x_j = \{0, \pi/4, \pi/2\}$ we have $h = \pi/4$ and

j	x_j	f_j
0	0	0
1	$\pi/4$	$1/\sqrt{2}$
2	$\pi/2$	1

So

$$\begin{aligned}\int_0^{\pi/2} \sin(x)\, dx &\simeq \frac{\pi/4}{2}(0+1) + \pi/4\left(\frac{1}{\sqrt{2}}\right)\\ &= \frac{\pi}{8}\left(1+\sqrt{2}\right) \simeq 0.948.\end{aligned}$$

Example

We look at

$$\int_0^{\pi/2} \sin(x)\,\mathrm{d}x$$

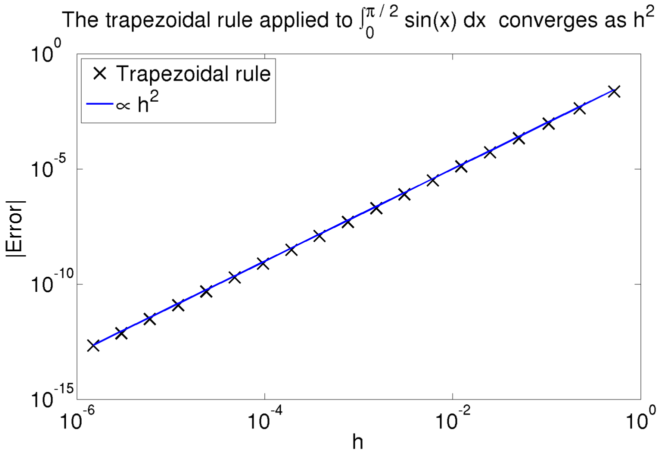
using the trapezoidal rule. The exact answer is 1.
With four points $x_j = \{0, \pi/6, \pi/3, \pi/2\}$ we have $h = \pi/6$ and

j	x_j	f_j
0	0	0
1	$\pi/6$	$1/2$
2	$\pi/3$	$\sqrt{3}/2$
3	$\pi/2$	1

So

$$\begin{aligned}\int_0^{\pi/2} \sin(x)\,\mathrm{d}x &\simeq \frac{\pi/6}{2}(0+1) + \pi/6\left(\tfrac{1}{2} + \tfrac{\sqrt{3}}{2}\right)\\ &= \tfrac{\pi}{12}\left(2+\sqrt{3}\right) \simeq 0.977.\end{aligned}$$

Example Convergence



The example converges as expected with resolution.

Summary

- When working with functions you have to decide how to represent the function using a finite amount of data.
- Quadrature (integration) based on
  - Evaluateing the function at a finite number of nodes;
  - Interpolating between nodes using a polynomial

  results in a Newton-Cotes formula/algorithm.
- The trapezoidal rule interpolates the function using a linear polynomial (straight line).
- The composite trapezoidal rule breaks the interval into many small pieces and applies the trapezoidal rule to each.
- If each subinterval has the same width h the trapezoidal rule is very simple and converges with an error $\propto h^2$.

Chapter 10 - Quadrature

G. Richardson

MATH3018/6141, Semester 1

Improving Convergence

The aim of numerical quadrature is to compute

$$\int_a^b f(x)\,\mathrm{d}x$$

where f is a real function of a single variable x.

Simple methods use polynomial quadrature: function values known at *nodes* are interpolated by a (piecewise) polynomial g. The quadrature is approximated by $\int g(x)\,\mathrm{d}x$.

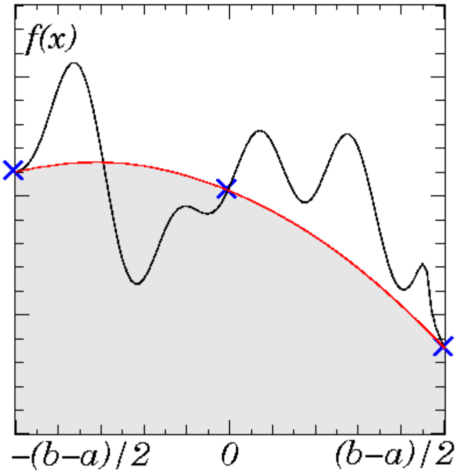
Introduced (composite) *trapezoidal* rule, where g is a (piecewise) linear polynomial – a straight line connecting neighbouring nodes.

Can improve accuracy by increasing the order of polynomial interpolation. Simplest improvement uses a quadratic polynomial, giving *Simpson's* rule.

Simpson's rule

Three points are needed to determine a unique second order polynomial (*i.e.* a quadratic). Our discussion of Simpson's rule assumes equally spaced points. On the single interval $[a, b]$ we use points $\{a, (a+b)/2, b\}$ with $h = (b-a)/2$.

Work in coordinates about the centre of the interval, $\hat{x} = x - (a+b)/2$. Hence $\hat{x} \in [-h, h]$.



Interpolating polynomial

In the coordinates $\hat{x} = x - (a+b)/2$ we have

$$\begin{aligned} g(\hat{x}) &= \alpha \hat{x}^2 + \beta \hat{x} + \gamma, \qquad \hat{x} \in [-h, h] \\ \Rightarrow \qquad A &= \int_{-h}^{h} g(\hat{x})\,\mathrm{d}x \\ &= \tfrac{\alpha}{3}\left(2h^3\right) + \gamma\,(2h)\,. \end{aligned}$$

Simple to evaluate the coefficients as

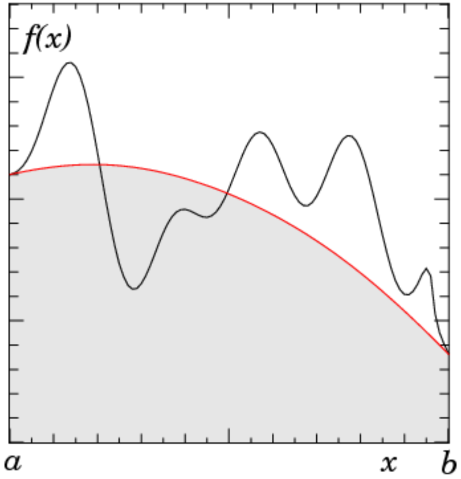
$$\begin{gathered} g(\hat{x}=0) = \gamma, \\ g(\hat{x}=-h) + g(\hat{x}=h) = 2\alpha h^2 + 2\gamma. \end{gathered}$$

Putting this together we have

$$\begin{aligned} A &= \tfrac{h}{3}\left(\left(2\alpha h^2 + 2\gamma\right) + 4\gamma\right) \\ &= \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]. \end{aligned}$$

Composite Simpson's Rule

As with trapezoidal rule, basic Simpson's rule is very inaccurate.



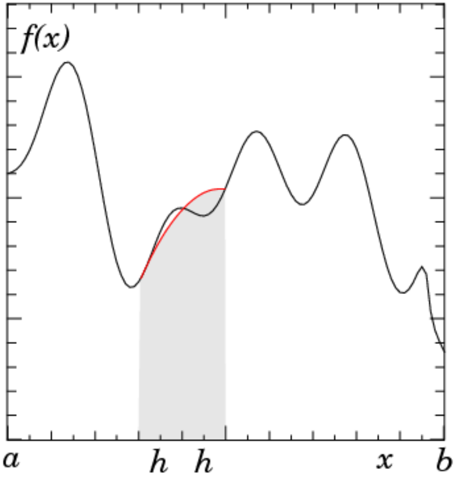
Composite Simpson's Rule

Instead use the *composite* Simpson's rule. Interval is divided into $N/2$ equal subintervals length $2h$,

$$h = (b - a)/N,$$

and Simpson's rule is applied to each subinterval.

The factor of 2 implies 3 nodes in each subinterval: the boundary points and a centre point. Each subinterval has a unique interpolating quadratic, and no quadratic overlaps, giving a simple piecewise polynomial.



Composite Simpson's rule formula

Use an evenly spaced grid with $N+1$ nodes (where N is even). Then Simpson approximates integral by

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x &\simeq \frac{h}{3}\left[f(a)+f(b)+4\sum_{j=1}^{N/2} f_{2j-1}+2\sum_{j=1}^{N/2-1} f_{2j}\right]\\ &= \frac{h}{3}\left[f(a)+4f(a+h)+2f(a+2h)+4f(a+3h)+\cdots\right.\\ &\qquad\qquad \left.+2f(b-2h)+4f(b-h)+f(b)\right].\end{aligned}$$

The global error from Simpson's rule is

$$\text{Error} \le \frac{M_4}{180}(b-a)h^4, \quad M_4 = \max|f^{(4)}(x)|.$$

Errors: 1

We find the error by bounding the error for a single interval. Write Simpson's rule as a power law expansion in terms of the width of the interval $2h$ with centre x_j, and compare to the exact solution expansion.

For one interval of Simpson's rule:

$$A_j = \frac{h}{3}\left[f(x_j - h) + 4f(x_j) + f(x_j + h)\right].$$

Taylor expand about the centre point of the interval x_j:

$$A_j = 2hf(x_j) + \frac{h^3}{3}f^{(2)}(x_j) + \frac{h^5}{36}f^{(4)}(x_j) + \mathcal{O}(h^6).$$

Errors: 2

As before we need the exact result in powers of h. Working about the centre of the interval we define

$$F(t) = \int_{x_j - t}^{x_j + t} f(x)\,\mathrm{d}x$$

so that

$$F(h) = \int_{x_j - h}^{x_j + h} f(x)\,\mathrm{d}x.$$

We note that

$$\frac{\mathrm{d}^n F}{\mathrm{d}t^n} = f^{(n-1)}(x_j + t) + (-1)^{n-1} f^{(n-1)}(x_j - t)$$

or in particular

$$\frac{\mathrm{d}F}{\mathrm{d}t} = f(x_j + t) + f(x_j - t).$$

Errors: 3

From this we can Taylor expand about $t = 0$ to find

$$F(h) \sim \left(F + h\,\frac{\mathrm{d}F}{\mathrm{d}t} + \frac{h^2}{2}\,\frac{\mathrm{d}^2 F}{\mathrm{d}t^2} + \frac{h^3}{6}\,\frac{\mathrm{d}^3 F}{\mathrm{d}t^3} + \frac{h^4}{24}\,\frac{\mathrm{d}^4 F}{\mathrm{d}t^4} + \frac{h^5}{120}\,\frac{\mathrm{d}^5 F}{\mathrm{d}t^5} \right)\Bigg|_{t=0}$$

But $F'(0) = 2f(x_j)$, $F''(0) = 0$, $F'''(0) = 2f''(x_j)$, $F''''(0) = 0$ and $F'''''(0) = 2f^{(4)}(x_j)$. It follows that

$$F(h) = 2hf(x_j) + \tfrac{h^3}{3} f^{(2)}(x_j) + \tfrac{h^5}{60} f^{(4)}(x_j) + \dots .$$

Comparing with Simpson's rule

$$A_j = 2hf(x_j) + \tfrac{h^3}{3} f^{(2)}(x_j) + \tfrac{h^5}{36} f^{(4)}(x_j) + \dots$$

the error in the subinterval is

$$|A_j - F(h)| \leq \tfrac{h^5}{90} |f^{(4)}(x_j)|.$$

Summing over all $N/2$ subintervals and using $hN = (b - a)$ gives

$$\text{Error} \leq \frac{(b-a)h^4}{180} M_4, \quad M_4 = \max_{x \in [a,b]} |f^{(4)}(x)|.$$

Example

We look at

$$\int_0^{\pi/2} \sin(x)\,dx$$

using Simpson's rule. The exact answer is 1.

With three points $x_j = \{0, \pi/4, \pi/2\}$ we have $h = \pi/4$ and

j	x_j	f_j
0	0	0
1	$\pi/4$	$1/\sqrt{2}$
2	$\pi/2$	1

So

$$\begin{aligned}\int_0^{\pi/2} \sin(x)\,dx &\simeq \frac{\pi/4}{3}(0+1) + \frac{4(\pi/4)}{3}\left(\tfrac{1}{\sqrt{2}}\right) \\ &= \tfrac{\pi}{12}\left(1+2\sqrt{2}\right) \\ &\simeq 1.002.\end{aligned}$$

Example

We look at

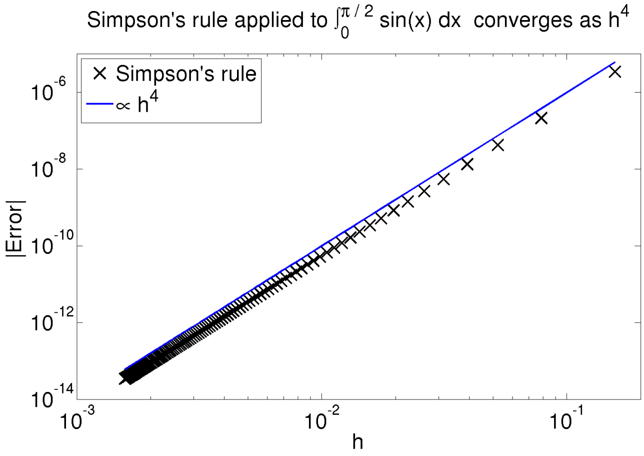
$$\int_0^{\pi/2} \sin(x)\,dx$$

using Simpson's rule. The exact answer is 1.
With five points $x_j = \{0, \pi/8, \pi/4, 3\pi/8, \pi/2\}$ we have $h = \pi/8$ and

j	x_j	f_j
0	0	0
1	$\pi/8$	0.38268
2	$\pi/4$	$1/\sqrt{2}$
3	$3\pi/8$	0.92388
4	$\pi/2$	1

$$\int_0^{\pi/2} \sin(x)\,dx \simeq \frac{\pi/8}{3}(0+1) + \frac{4(\pi/8)}{3}(0.38268 + 0.92388) + \frac{2(\pi/8)}{3}\left(1/\sqrt{2}\right) \quad \simeq 1.000135.$$

Example Convergence



The example converges as expected with resolution.

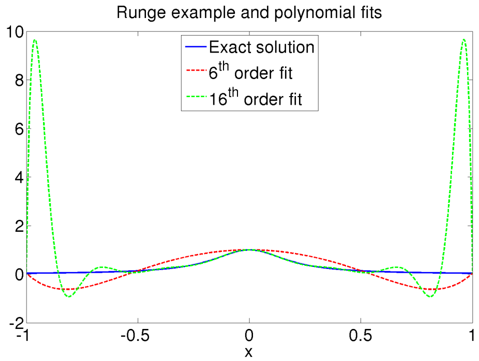
Yet higher order

Why not use cubics, quartics, . . . ?

One way of deriving *spectral* methods, which converge faster than any polynomial.

However, care is required. With equally spaced nodes may see problems, e.g. the *Runge example*

$$f(x) = \frac{1}{1 + 25x^2}.$$



Richardson Extrapolation

A general trick for *any* numerical method with “enough” information about the error. Illustrate with Simpson’s rule.

Assume the exact solution is I and Simpson’s rule gives I_h for subintervals width h. We know

$$I - I_h \simeq Ch^4.$$

Assume equality and that C is independent of h. Hence

$$\begin{aligned} I - I_h &= Ch^4 \\ I - I_{2h} &= C(2h)^4 \\ \Rightarrow \qquad I &= \frac{2^4 I_h - I_{2h}}{2^4 - 1}. \end{aligned}$$

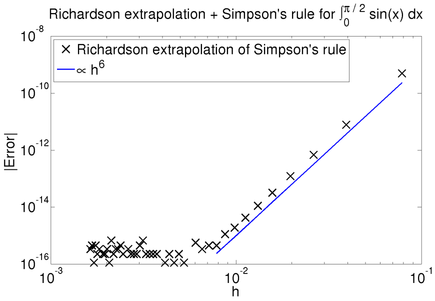
Example of Richardson Extrapolation

Richardson Extrapolation removes the leading order error term (does not give exact answer). For Simpson's rule this leads to convergence $\propto h^6$.

Richardson extrapolation can fail if

- function under-resolved, or
- error behaves badly (e.g. f has limited differentiability)

For this reason normally used to estimate error, not to improve accuracy.



Summary

- Simpson's rule is a second order Newton-Cotes formula.
- The composite formula is usually employed using equally spaced nodes.
- The error converges as $\mathcal{O}(h^4)$.
- Richardson extrapolation takes advantage of prior knowledge about *rate* of convergence to eliminate the dominant error term.

Chapter 11 - Initial Value Problems

G. Richardson

MATH3018/6141, Semester 1

Ordinary Differential Equations

Consider first order systems of ordinary differential equations (ODEs), written

$$\boldsymbol{y}'(x) = \boldsymbol{f}(x, \boldsymbol{y}(x)).$$

Solution: any function $\boldsymbol{y}(x)$ that satisfies the equation.

There are many possible solutions; for a *unique* solution need to specify n independent conditions (for system size n).

If the n conditions are all specified at one point, *e.g.* $\boldsymbol{y}(0) = \boldsymbol{y}_0$, this is called an *initial value problem*.

Initial value problems are nearly always well-behaved, *i.e.* they have a unique solution.

Boundary value problems sometimes do not have unique solutions - even with n independent conditions specified.

Getting to the first order form

The ODE need not be written in first order form, e.g.

$$z'' + z' + 1 = 0, \quad z(0) = a,\ z'(0) = b.$$

Recast the problem using *auxilliary variables*, such as $w = z'$.

$$\begin{pmatrix} w \\ z \end{pmatrix}' = \begin{pmatrix} -w - 1 \\ w \end{pmatrix}, \quad z(0) = a,\ w(0) = b.$$

Not always best to use simple derivatives as auxilliary variables.

$$\text{e.g.} \quad \text{in} \quad \frac{1}{r^2}\left(r^2 P'/\rho\right)' = -4\pi\rho \quad \text{write} \quad \Pi(r) = r^2 P'/\rho$$

to transform the equation to the 1st order system

$$\begin{pmatrix} P \\ \Pi \end{pmatrix}' = \begin{pmatrix} \rho\Pi/r^2 \\ -4\pi r^2 \rho \end{pmatrix}.$$

Numerical differentiation

If $f(x)$ known at the nodes $\{x_j\}$ can compute the derivative from the nodal values.

Simple example: assume $f(x)$ is known at x_0 and $x_0 \pm h$ for some small h.

Fit the linear polynomial $g(x)$ (a straight line) to $f(x)$ with $x \in [x_0, x_0 + h)$: approximate the derivative by slope of the line

$$f'(x_0) \simeq g'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}.$$

This is the *forward difference* estimate of the derivative.

Similarly, a linear fit in $(x_0 - h, x_0]$ gives

$$f'(x_0) \simeq g'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h}.$$

This is the *backward difference* estimate of the derivative.

Numerical differentiation - accuracy

$$f'(x_0) \simeq g'(x_0) = \frac{f(x_0+h) - f(x_0)}{h}.$$

The accuracy of the approximation depends on the choice of interpolating polynomial.

Check accuracy using Taylor expansion:

$$\begin{aligned} f(x_0+h) &= f(x_0) + hf'(x_0) + \tfrac{h^2}{2!}f''(x_0) + \mathcal{O}(h^3) \\ \Rightarrow \quad \frac{f(x_0+h) - f(x_0)}{h} &= \frac{f(x_0) + hf'(x_0) + \tfrac{h^2}{2!}f''(x_0) + \mathcal{O}(h^3) - f(x_0)}{h} \\ &= f'(x_0) + \mathcal{O}(h). \end{aligned}$$

The same accuracy result holds for

$$f'(x_0) = \frac{f(x_0) - f(x_0-h)}{h}.$$

Example

Compute the derivative of $f(x) = e^x$ at $x = 0$ using forward differencing,

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h},$$

checking the error to the exact result $f'(0) \equiv 1$.

h	Formula	$f'_h(0)$	e_h	Error ratio
0.5	$\frac{\exp(0.5)-\exp(0)}{0.5}$	1.297	0.297	–
0.05	$\frac{\exp(0.05)-\exp(0)}{0.05}$	1.0254	0.0254	0.085
0.005	$\frac{\exp(0.005)-\exp(0)}{0.005}$	1.00250	0.0025	0.098

The error ratio is (asymptotically) the same as the step-size ratio.

Central differencing

For more accuracy use a quadratic through the points $\{x_0 - h, x_0, x_0 + h\}$:

$$f(x) \simeq g(x) = \alpha(x - x_0)^2 + \beta(x - x_0) + \gamma.$$

Compute the derivative at x_0:

$$f'(x_0) = \beta = \frac{f(x_0 + h) - f(x_0 - h)}{2h}.$$

This *central differencing* estimate is more accurate than forward/backward differencing:

$$\frac{f(x_0 + h) - f(x_0 - h)}{2h} = f'(x_0) + \mathcal{O}(h^2).$$

Example

Compute the derivative of $f(x) = e^x$ at $x = 0$ with central differencing,

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h},$$

checking the error to the exact result $f'(0) \equiv 1$.

h	Formula	$f'_h(0)$	Error ratio
0.5	$\frac{\exp(0.5) - \exp(-0.5)}{1}$	1.042	–
0.05	$\frac{\exp(0.05) - \exp(-0.05)}{0.1}$	1.0004167	0.00988
0.005	$\frac{\exp(0.005) - \exp(-0.005)}{0.01}$	1.0000004180	0.0100

Central differencing is much more accurate, and the error ratio is (asymptotically) the *square* of the step-size ratio.

Other numerical differencing formulas

- Intuitively, use of information from both sides of x_0 leads to higher accuracy (e.g. central differencing).
- Can build differencing formulas directly from Taylor series without using an interpolating function.
- Estimates of higher derivatives can be made using either an interpolation function or repeated numerical differencing. For example, estimate $f''(x)$ using forward and backward differencing:

$$\begin{aligned} f''(x_0) &\simeq \frac{f'(x_0) - f'(x_0 - h)}{h} \\ &\simeq \frac{(f(x_0+h) - f(x_0)) - (f(x_0) - f(x_0-h))}{h^2} \\ &= \frac{f(x_0+h) + f(x_0-h) - 2f(x_0)}{h^2}. \end{aligned}$$

Euler's method

Using differencing construct the simplest method for solving

$$\boldsymbol{y}'(x) = \boldsymbol{f}(x, \boldsymbol{y}(x)), \quad \boldsymbol{y}(0) = \boldsymbol{y}_0.$$

Using step size h, apply forward differencing to $\boldsymbol{y}$:

$$\frac{\boldsymbol{y}_{n+1} - \boldsymbol{y}_n}{h} = \boldsymbol{f}(x_n, \boldsymbol{y}_n) \quad \Rightarrow \quad \boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(x_n, \boldsymbol{y}_n).$$

Given initial data $\boldsymbol{y}_0$, for each stage we can compute $\boldsymbol{y}_{n+1}$ from data $\boldsymbol{y}_n$. Methods of this sort are called *explicit*.

Compute the error using Taylor expansion:

$$\boldsymbol{y}_{n+1} \equiv \boldsymbol{y}\left(x_0 + (n+1)h\right) = \boldsymbol{y}_n + h\boldsymbol{f}(x_n, \boldsymbol{y}_n) + \mathcal{O}(h^2).$$

This means the *local* error is $\mathcal{O}(h^2)$. However the *global* error is $\mathcal{O}(h)$.

Example

Apply Euler's method to

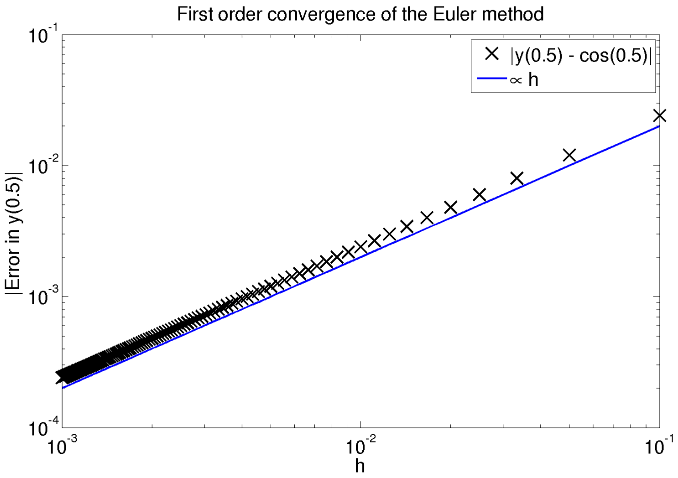
$$y'(x) = -\sin(x), \quad y(0) = 1$$

up to $x = 0.5$. Using step size $h = 0.1$ gives

n	x_n	y_n	$f(x_n, y_n) = -\sin(x_n)$	$\cos(x_n)$
0	0.0	1.00	0.000	1.000
1	0.1	1.00	-0.100	0.995
2	0.2	0.99	-0.199	0.980
3	0.3	0.97	-0.296	0.955
4	0.4	0.94	-0.389	0.921
5	0.5	0.90		0.878

The error is 2.4%; using $h = 0.01$ the final result is $y_{50} = 0.880$, an error of 0.24%: first order convergence.

Convergence



More conclusive evidence that the convergence of Euler's method is first order: the global error is $\propto h$.

Example: 2

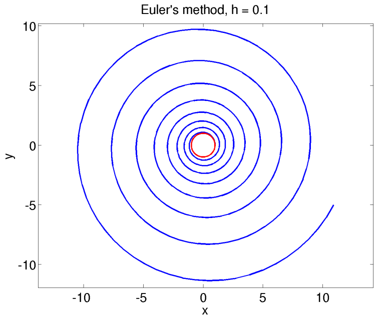
Consider the system

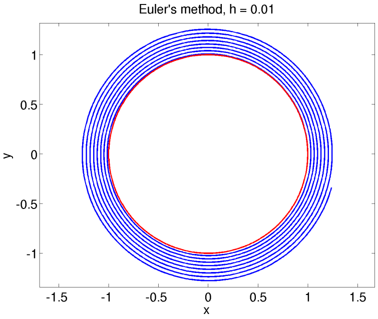
$$\begin{cases} \dot{x} = -y \\ \dot{y} = x \end{cases}, \quad x(0) = 1, \ y(0) = 0.$$

In polar coordinates this is $\dot{r} = 0$, $\dot{\phi} = 1$.

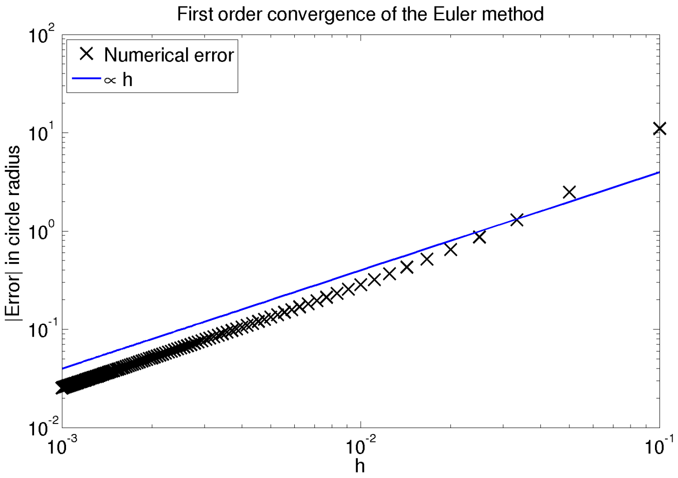
Use Euler's method with $h = 0.1$. At $t = 50$ the result is slightly out of phase and the radius has grown significantly.

With $h = 0.01$ the errors are smaller, but still visible.





Convergence: 2



More conclusive evidence that the convergence of Euler's method is (asymptotically) first order: the global error is $\propto h$.

Summary

- Initial value problems are ODEs specified by an initial condition $\boldsymbol{y}(0) = \boldsymbol{y}_0$.
- Numerical differentiation using finite differences can easily produce
  - first order accurate approximations to $f'(x)$ using *forward* or *backward* differencing, or
  - second order accurate approximations to $f'(x)$ using central differencing.
- General central differencing about x_i uses the closest points to x_i in a symmetric fashion to approximate whichever derivative is required.
- To prove the convergence behaviour Taylor's theorem is used.
- Approximations to higher derivatives are also easy to compute.
- Using forward differencing in the IVP gives Euler's method.
- Euler's method has a local error $\propto h^2$, and a global error $\propto h$.

Chapter 12 - Initial Value Problems

G. Richardson

MATH3018/6141, Semester 1

Geometrical interpretation of Euler's method

Considering IVPs in the form

$$\boldsymbol{y}'(x) = \boldsymbol{f}(x, \boldsymbol{y}(x)).$$

The simple (first order accurate, explicit) Euler method is

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(x_n, \boldsymbol{y}_n).$$

Euler's method is not sufficiently accurate for practical use. The geometrical interpretation is that the slope ($\boldsymbol{y}'$) over the interval $[x_n, x_{n+1}]$ is approximated by the slope at the *beginning* of the interval,

$$\boldsymbol{y}' \simeq \boldsymbol{y}'(x_n).$$

Try a better approximation to the slope for a more accurate result.

Predicting and correcting

Still using forward differencing for $\boldsymbol{y}'$, an “average” slope is

$$\frac{\boldsymbol{y}_{n+1} - \boldsymbol{y}_n}{h} = \frac{\boldsymbol{f}(x_n, \boldsymbol{y}_n) + \boldsymbol{f}(x_{n+1}, \boldsymbol{y}_{n+1})}{2}.$$

Rearrange: gives the unknown $\boldsymbol{y}_{n+1}$ as

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{h}{2}\left(\boldsymbol{f}(x_n, \boldsymbol{y}_n) + \boldsymbol{f}(x_{n+1}, \boldsymbol{y}_{n+1})\right).$$

But we need $\boldsymbol{y}_{n+1}$ to compute the right-hand-side.

Predicting and correcting

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{h}{2}\left(\boldsymbol{f}(x_n, \boldsymbol{y}_n) + \boldsymbol{f}(x_{n+1}, \boldsymbol{y}_{n+1})\right).$$

Instead compute a *guess* for $\boldsymbol{y}_{n+1}$, and use this guess when required. This is the *predictor* step:

$$\boldsymbol{y}^{(p)}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(x_n, \boldsymbol{y}_n)$$

if we use Euler's method for the prediction.

Using this, apply the *corrector* step

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{h}{2}\left(\boldsymbol{f}(x_n, \boldsymbol{y}_n) + \boldsymbol{f}(x_{n+1}, \boldsymbol{y}^{(p)}_{n+1})\right),$$

which describes the (Euler) *predictor-corrector* method.

The *local* error is $\propto \mathcal{O}(h^3)$.

Example

Apply the Euler predictor-corrector method to

$$y'(x) = -\sin(x), \quad y(0) = 1$$

to integrate up to $x = 0.5$. With $h = 0.1$:

n	x_n	y_n	$f(x_n, y_n)$	$y_{n+1}^{(p)}$	$f(x_{n+1}, y_{n+1}^{(p)})$	$\cos(x_n)$
0	0.0	1.000	0.000	1.000	-0.100	1.000
1	0.1	0.995	-0.100	0.985	-0.199	0.995
2	0.2	0.980	-0.199	0.960	-0.296	0.980
3	0.3	0.955	-0.296	0.926	-0.389	0.955
4	0.4	0.921	-0.389	0.882	-0.479	0.921
5	0.5	0.878				0.878

The error is 0.1%, not visible at this precision. With $h = 0.01$ then the error is 0.001%, showing second order convergence.

Example: 2

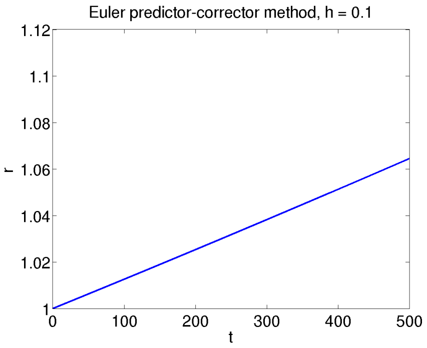
Consider the system

$$\begin{cases} \dot{x} = -y \\ \dot{y} = x \end{cases}, \quad x(0) = 1, \; y(0) = 0.$$

In polar coordinates this is $\dot{r} = 0$, $\dot{\phi} = 1$.

Use the predictor-corrector method with $h = 0.1$. At $t = 500$ the result matches the correct answer to the eye.

The growth of the radius makes the errors visible.



Example: 2

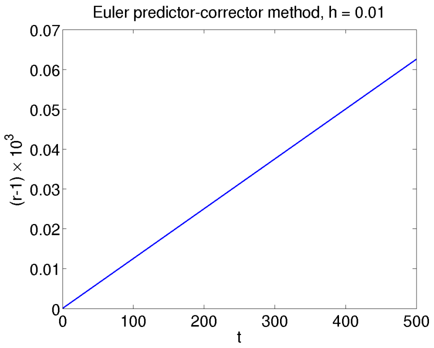
Consider the system

$$\begin{cases} \dot{x} = -y \\ \dot{y} = x \end{cases}, \quad x(0) = 1, \ y(0) = 0.$$

In polar coordinates this is $\dot{r} = 0$, $\dot{\phi} = 1$.

Use the predictor-corrector method with $h = 0.01$. At $t = 500$ the result matches the correct answer to the eye.

Looking at the growth of the radius makes the errors visible even now.

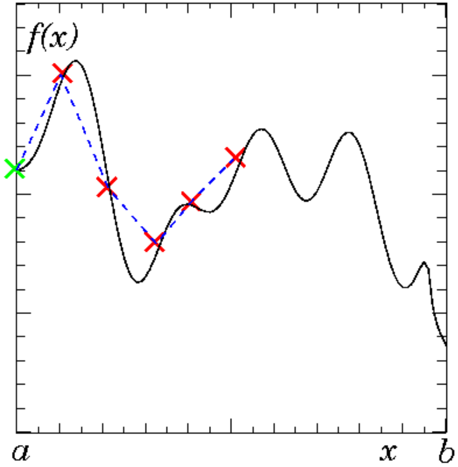


Errors

With quadrature using e.g. Simpson's rule the *local* error on any subinterval gave

$$|\mathcal{E}_{\text{global}}| \leq \sum |\mathcal{E}_{\text{local}}|.$$

For an IVP we still have the *local* and *global* errors. But steps are not independent so the error bound is different.



Truncation vs. roundoff

All error (local and global) has two pieces: *truncation* and *roundoff* error.

1. Roundoff: inherent in the floating point representation. Usually extremely small, it cannot be controlled.
2. Truncation: the result of replacing the continuous differential equation with a discrete algorithm. Usually dominant. Typically depends on “step size” h which can be controlled.

The *total error* (to minimize) is the sum of the global truncation and global roundoff errors.

We bound the global truncation error only as we expect roundoff error to be negligible.

The Global error theorem

Global Error Theorem: If the local truncation error for a 1st order ODE is bounded by

$$|\mathcal{E}_{\text{local,truncation}}| \leq \delta \propto h^{p+1},$$

then the global truncation error after n steps, of length h, is bounded by

$$|\mathcal{E}_{\text{global,truncation}}| \leq \left| \delta \frac{e^{n\lambda h} - 1}{e^{\lambda h} - 1} \right|.$$

where λ is a bound on derivative of f such that $|\partial f/\partial y| \leq \lambda$.

In practice can normally *assume* that $\lambda h \ll 1$.

If we also solve in $x \in (a, b]$ then $nh = (b - a)$.

Taylor expanding the error bound we have

$$|\mathcal{E}_{\text{Global}}| \leq \left| \delta \frac{(e^{\lambda(b-a)} - 1)}{\lambda h + \mathcal{O}(h^2)} \right| \leq \mathcal{O}(h^p) \left| e^{\lambda(b-a)-1} \right|.$$

Runge-Kutta methods

Runge-Kutta methods are based on Taylor's theorem to ensure the desired accuracy.

1. Consider a single step from known data $y_n(x_n)$.
2. From known data compute estimate k_1 for $y'(x_n)$
$$k_1 = hf(x_n, y_n)$$
3. Estimate y at $x_n + \alpha h$ by writing
$$y^{(1)} = x_n + \alpha k_1$$
.
4. Compute estimate (k_2) for y' at $x_n + \alpha h$ by writing
$$k_2 = hf(x_n + h\alpha, y_n + h\alpha k_1).$$
5. Repeat as many times as necessary estimating y and hf at different points.
6. Finally combine the result to obtain $y_{n+1} = h(ak_1 + bk_2 + \dots)$.

Runge-Kutta methods

Such methods are called *multistage*:

- a number of estimates of f are combined to improve accuracy;
- only the previous value y_n is required to start the algorithm.

To derive $a, b, \dots, \alpha, \dots$ Taylor expand the algorithm about $x = x_n$ and match to Taylor expansion of exact solution about $x = x_n$ using the ODE.

Example: RK2

For the second order method we have

$$\begin{aligned} y_{n+1} &= y_n + (ak_1 + bk_2), \\ k_1 &= hf(x_n, y_n), \\ k_2 &= hf(x_n + h\alpha, y_n + h\alpha k_1). \end{aligned}$$

We have three free parameters a, b, α to fix, although it is immediately clear that if the scheme is to approximate $y' = f(x, y)$ in the limit $h \to 0$ then $a + b = 1$.

To assess the local error we first Taylor expand $y(x_n + h)$ about x_n:

$$\begin{aligned} y(x_n + h) &= y(x_n) + hy'(x_n) + \tfrac{h^2}{2} y''(x_n) + \dots \\ &= y(x_n) + hf(x_n, y(x_n)) + \tfrac{h^2}{2} \left(\tfrac{\partial}{\partial x} f(x, y(x)) \Big|_{x=x_n} \right) + \dots \\ &= y(x_n) + hf(x_n, y(x_n)) + \tfrac{h^2}{2} \left(\partial_x f(x_n, y(x_n)) + y'(x_n) \partial_y f(x_n, y(x_n)) \right) + \dots \end{aligned}$$

where we have substitute $y'(x_n) = f(x_n, y(x_n))$

Example: RK2

Translate this into the algortimic notation by writing

$y_n = y(x_n),\ f_n = f(x_n, y(x_n)),\ (\partial_x f)_n = \partial_x f(x_n, y(x_n)),\ (\partial_y f)_n = \partial_y f(x_n, y(x$

in the Taylor expansion of $y(x_n + h)$ to find

$$y(x_n + h) = y_n + hf_n + \frac{h^2}{2}\left((\partial_x f)_n + f_n(\partial_y f)_n\right) + O(h^3)$$

Now Taylor expand the Algorithm about $x = x_n$

$$\begin{aligned} y_{n+1} &= y_n + ak_1 + bk_2 \\ &= y_n + ahf(x_n, y_n) + bhf(x_n + \alpha h, y_n + \alpha hf_n) \\ &= y_n + ahf_n + hb\left(f_n + \alpha h(\partial_x f)_n + \alpha hf_n(\partial_y f)_n\right) + O(h^3) \\ &= y_n + (a+b)hf_n + \alpha bh^2\left((\partial_x f)_n + f_n(\partial_y f)_n\right) + O(h^3) \end{aligned}$$

Compare this against Taylor expansion of $y(x_n + h)$ above. Get local error to be $O(h^3)$ if we choose

$$\begin{cases} a + b = 1 \\ \quad \alpha b = 1/2 \end{cases}.$$

Example: RK2 (II)

The RK2 method

$$\begin{aligned} y_{n+1} &= y_n + ak_1 + bk_2, \\ k_1 &= hf(x_n, y_n), \\ k_2 &= hf(x_n + \alpha h, y_n + \beta k_1) \end{aligned}$$

with coefficients

$$\begin{cases} a + b = 1 \\ \quad \alpha b = 1/2 \end{cases}$$

is not completely specified; there is essentially one free parameter.

Not all choices are stable. The classic choice is $a = 1/2 = b$, $\alpha = 1$: this is the Euler predictor-corrector method.

Runge-Kutta 4

Most widely used Runge-Kutta method is the classic fourth order one. This requires fixing eight free parameters by matching to order h^4. This gives a family of methods again.

Standard choice is

$$\begin{aligned} y_{n+1} &= y_n + \tfrac{1}{6}\left(k_1 + 2(k_2 + k_3) + k_4\right), \\ k_1 &= hf(x_n, y_n), \\ k_2 &= hf(x_n + h/2, y_n + k_1/2), \\ k_3 &= hf(x_n + h/2, y_n + k_2/2), \\ k_4 &= hf(x_n + h, y_n + k_3). \end{aligned}$$

The local error term is $\mathcal{O}(h^5)$ leading to a global error $\mathcal{O}(h^4)$.

Example

Apply the RK4 method to

$$y'(x) = -\sin(x), \quad y(0) = 1.$$

Integrate to $x = 0.5$. Using $h = 0.1$ gives an error $4.8 \times 10^{-7}\%$; using $h = 0.01$ gives an error of $4.8 \times 10^{-11}\%$, showing fourth order convergence.

Compare with an error, for $h = 0.01$, of $10^{-3}\%$ for the Euler predictor-corrector method, and 0.24% for the simple Euler method.

RK4 more efficient *despite* needing four times the function evaluations of Euler's method.

Example: 2

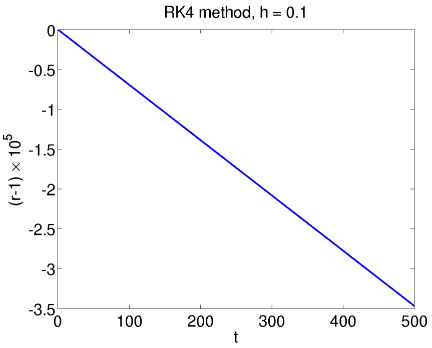
Consider the system

$$\begin{cases} \dot{x} = -y \\ \dot{y} = x \end{cases}, \quad x(0) = 1, \; y(0) = 0.$$

In polar coordinates this is $\dot{r} = 0$, $\dot{\phi} = 1$.

Use the RK4 method with $h = 0.1$. At $t = 500$ the result matches the correct answer to the eye.

The growth of the radius makes the errors visible, but they are still tiny.



Example: 2

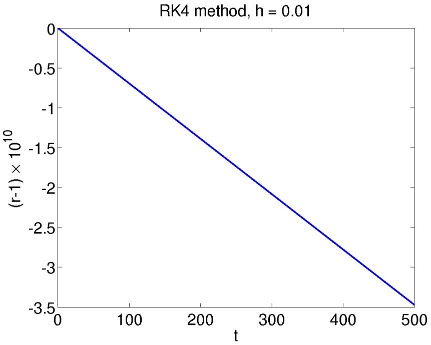
Consider the system

$$\begin{cases} \dot{x} = -y \\ \dot{y} = x \end{cases}, \quad x(0) = 1,\; y(0) = 0.$$

In polar coordinates this is $\dot{r} = 0$, $\dot{\phi} = 1$.

Use the RK4 method with $h = 0.01$. At $t = 500$ the result matches the correct answer to the eye.

The growth of the radius remains, but is minute.



Summary

- The errors at each step in an IVP solver are not independent. Hence the total error is not bounded by the sum of the local errors.
- However where the local error is $\mathcal{O}(h^{n+1})$ the global error is $\mathcal{O}(h^n)$. Such a method is said to be of *order n*.
- Euler's method has local error $\propto h^2$ and hence global error $\propto h$.
- The Euler predictor-corrector method has local error $\propto h^3$ and hence global error $\propto h^2$.
- *Multistage* methods such as Runge-Kutta methods require only one known value $\boldsymbol{y}_n$ to start, and compute (many) estimates of the function $\boldsymbol{f}$ for the algorithm to update $\boldsymbol{y}_{n+1}$.
- Runge-Kutta methods are the classic multistage methods; the predictor-corrector method is a second order RK method.
- RK4 is very widely used in practice becuase of its accuracy (global error $\mathcal{O}(h^4)$), ease of use and stability.