

Numerical Analysis Comprehensive Exam

Notes on Linear Algebra and Direct Solutions to Linear Systems

1. Material on the past six years of exams not covered in these notes:
 - a. QR decomposition
 - b. eigenvalues and eigenvectors
 - c. singular value decomposition

2. Linear Systems: $Ax = b$
Is the vector b some combination of the columns of A ?
 - a. Square (# rows = # columns)
 - b. Overdetermined (# rows > # columns)
 - c. Underdetermined (# rows < # columns)

3. LU Decomposition: $A = LU$
 - a. L is lower triangular, U is upper triangular
 - b. Gaussian elimination
 - i. Perform matrix operations to turn A into an upper triangular matrix
 - ii. For each column i , zero out all entries below the diagonal by subtracting multiples of row i from all rows below it

$$A = \begin{bmatrix} 2 & 1 & 2 \\ 2 & 3 & 7 \\ 4 & 10 & 1 \end{bmatrix}$$

1. Subtract $\begin{bmatrix} 2 & 1 & 2 \end{bmatrix}$ from row 2

2. Subtract $2 \cdot \begin{bmatrix} 2 & 1 & 2 \end{bmatrix}$ from row 3

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix} A = \begin{bmatrix} 2 & 1 & 2 \\ 2-2 \cdot 1 & 3-1 \cdot 1 & 7-2 \cdot 1 \\ 4-2 \cdot 2 & 10-1 \cdot 2 & 1-2 \cdot 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix} A = \begin{bmatrix} 2 & 1 & 2 \\ 0 & 2 & 5 \\ 0 & 8 & -3 \end{bmatrix}$$

3. Subtract $4 \cdot \begin{bmatrix} 0 & 2 & 5 \end{bmatrix}$ from row 3

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix} A = \begin{bmatrix} 2 & 1 & 2 \\ 0 & 2 & 5 \\ 0 & 8-2 \cdot 4 & -3-5 \cdot 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & -4 & 1 \end{bmatrix} A = \begin{bmatrix} 2 & 1 & 2 \\ 0 & 2 & 5 \\ 0 & 0 & -23 \end{bmatrix}$$

4. Move terms to isolate A

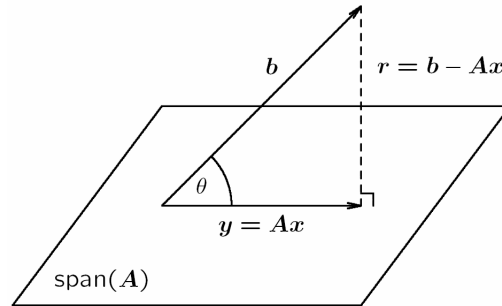
$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 4 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 2 \\ 0 & 2 & 5 \\ 0 & 0 & -23 \end{bmatrix}$$

4. Linear Least Squares: $\min_x \|Ax - b\|_2^2$

Find the vector x that most closely satisfies $Ax = b$

- a. For solving square or overdetermined systems
- b. Has a closed form solution: $A^T Ax = A^T b$
- c. Proofs

i. Geometric



choose x so that r is smallest

$$r = Ax - b \perp A$$

$$A^T (Ax - b) = 0$$

$$A^T Ax = A^T b$$

ii. Algebraic

$$\min_x \|Ax - b\|_2^2 = \min_x f(x)$$

$$\text{set } \nabla f(x) = 0$$

$$f(x) = (Ax - b)^T (Ax - b)$$

$$\nabla f(x) = 2(Ax - b)^T A = 2(A^T Ax - A^T b) = 0$$

d. Fall 2004, Problem 2

1 Numerical Integration

1.1 Numerical Quadrature

Suppose we want to compute:

$$I(f) = \int_a^b f(x)dx. \quad (1)$$

An n point quadrature formula has the form

$$I(f) = \int_a^b f(x)dx = \sum_{i=1}^n w_i f(x_i) + R_n. \quad (2)$$

The points x_i in which the function f is evaluated are called **nodes**, the multipliers w_i are called the weights, and R_n is the remainder or error. To estimate the value of the integral, we simply compute the sum

$$I(f) \approx \sum_{i=1}^n w_i f_i \quad (3)$$

which is known as a quadrature rule.

The error term R_n can be estimated by means of a Taylor series expansion of the integrand function, as we will see later.

x_i and w_i are based on polynomial interpolation. Different configurations of them lead to different rules as we will discuss later. The basic idea is the following. The integrand function f is sampled at some number of points (nodes), the polynomial that interpolates the function at those points is determined, and the integral of the interpolant is then taken as an approximation to the integral of the original function. Finally, the weights are computed according to the integral.

1.2 Newton Cotes Quadrature

If the nodes x_i are equally spaced in the interval $[a, b]$, the resulting interpolatory quadrature rule is known as a **Newton-Cotes quadrature**.

$n = 0, 1, 2$ are three special Newton-Cotes quadratures that should be memorized.

- $n = 0$, the interpolant is a constant which gives a one-point quadrature rule known as the mid-point rule or rectangle rule:

$$I(f) = M(f) = (b - a)f\left(\frac{a + b}{2}\right) \quad (4)$$

- $n = 1$, the interpolant is a straight line which gives a two-point quadrature rule known as the trapezoid rule:

$$I(f) = M(f) = \frac{(b - a)}{2} (f(a) + f(b)) \quad (5)$$

- $n = 2$, the interpolant is a quadratic which gives a three-point quadrature rule known as **Simpson's rule**:

$$I(f) = M(f) = \frac{b - a}{6} \left(f(a) + 4f\left(\frac{a + b}{2}\right) + f(b) \right). \quad (6)$$

1.3 Method of Undertermined Coefficients

Suppose we want to determine the weights of the trapezoid rule

$$I(f) = w_1 f(a) + w_2 f(b). \quad (7)$$

Take $f(x) = 1$ and $f(x) = x$ respectively and do the integrations. we get two equations.

$$rw_1 + w_2 = b - a \quad (8)$$

$$aw_1 + bw_2 = \frac{b^2 - a^2}{2} \quad (9)$$

After solving the linear equation, we get $w_1 = w_2 = \frac{b-a}{2}$.

1.4 Error Estimation

The error in the midpoint quadrature rule can be estimated by means of a Taylor series expansion about the midpoint $m = \frac{(a+b)}{2}$ of the interval $[a, b]$:

$$f(x) = f(m) + f'(m)(x - m) + \frac{f''(m)}{2}(x - m)^2 + \frac{f'''(m)}{6}(x - m)^3 + \frac{f^{iv}(m)}{24}(x - m)^4 + \dots \quad (10)$$

When we integrate this expression from a to b , the odd terms drop out, yielding

$$I(f) = f(m)(b - a) + \frac{f'''(m)}{24}(b - a)^3 + \frac{f^{iv}(m)}{1920}(b - a)^5 + \dots \quad (11)$$

$$= M(f) + E + F + \dots \quad (12)$$

where $E = \frac{f'''(m)}{24}(b - a)^3$ and $F = \frac{f^{iv}(m)}{1920}(b - a)^5$.

Similarly, for trapezoid rule, we have

$$I(f) = T(f) - 2E - 4F - \dots \quad (13)$$

Finally, we can combine the midpoint rule and trapezoid rule into simpson's error estimation as

$$I(f) = \frac{2}{3}(M(f) + E + F + \dots) + \frac{1}{3}(T(f) - 2E - 4F - \dots) \quad (14)$$

$$= \frac{2}{3}M(f) + \frac{1}{3}T(f) - \frac{2}{3}F + \dots \quad (15)$$

$$= S(f) - \frac{2}{3}F + \dots \quad (16)$$

1.5 Composite Quadrature Rules

It is not feasible to use arbitrarily high-order quadrature rules in an attempt to attain arbitrarily high accuracy in evaluating an integral over a given interval. A much better approach

We partition the interval $[a, b]$ into n panels, $[x_{i-1}, x_i]$, $i = 1, \dots, n$, with $a = x_0 < x_1 < \dots < x_n = b$, then the composite midpoint rule is given by

$$I(f) \approx M(f) = \sum_{i=1}^n (x_i - x_{i-1}) f\left(\frac{x_{i-1} + x_i}{2}\right), \quad (17)$$

and then composite trapezoid rule by

$$I(f) \approx T(f) = \sum_{i=1}^n (x_i - x_{i-1}) \frac{f(x_{i-1}) + f(x_i)}{2}, \quad (18)$$

finally the composite simpson's rule is given by

$$I(f) \approx S(f) = \sum_{i=1}^n (x_i - x_{i-1}) \frac{f(x_{i-1}) + 4f(x_{i-1} + x_i) + f(x_i)}{6}, \quad (19)$$

The dominant term in the remainder for the composite midpoint or trapezoid rules is $O(h^2)$ and the dominant term in the remainder for the composite simpson rule is $O(h^4)$, where $h = \frac{b-a}{n}$.

Talk a little bit about Simpson's rule.

2 Newton Method for solving non-linear equation

2.1 Iterative methods and Convergence Rate.

For a non-linear equation

$$f(x) = 0 \quad (20)$$

- Iterative methods start from a initial value x_0 and iteratively refine the value as $x_{k+1} = x_k + \Delta x_k$, where x_k is computed by some rules.
- Convergence rate. let $e_k = x_k - x^*$. if

$$\lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k|^r} = C. \quad (21)$$

then we so this method converges with rate r .

2.2 Newton's method

the formula of Newton method is given by

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \quad (22)$$

To prove Newton method is quadratic convergent, we compute

$$x_{k+1} - x^* = x_k - x^* - \frac{f(x_k)}{f'(x_k)} \quad (23)$$

$$= x_k - x^* - \frac{(x_k - x^*)f'(x_*) + \frac{(x_k - x^*)^2}{2}f''(x^*) + O((x_k - x^*)^3)}{f'(x^*) + f''(x^*)(x_k - x^*) + O((x_k - x^*)^2)} \quad (24)$$

$$= \frac{f''(x^*)}{2f'(x^*)}(x_k - x^*)^2 + O((x_k - x^*)^3). \quad (25)$$

Numerical Analysis, ODE's

1 First Order Differential Equations

An explicit, first order differential equation can be expressed by the following equation:

$$y' = f(y, t).$$

There are many approaches to solving such equations numerically, and three of them are considered here. Initial conditions are provided for $y(t_0) = y_0$, and the value of y is to be determined at $y(t_n)$, possibly with additional values in between. The general idea behind each is to divide up the equation along time and use an update rule to get an estimate y_{n+1} for $y(t_{n+1})$ given an estimate y_n for $y(t_n)$. The time step is $h = t_{n+1} - t_n$. There are several ways to compute an estimate for y_{n+1} .

2 Forwarded Euler

The simplest scheme is called Forward Euler, and the update rule is:

$$y_{n+1} = y_n + hf(y_n, t_n).$$

The motivation behind Forward Euler is to assume that the solution is approximately linear. Use the differential equation to obtain the slope of the differential equation at the current point, and advance along the line with that slope, as illustrated in Figure 1.

2.1 Derivation

$$y(t_{n+1}) = y(t_n) + y'(t_n)(t_{n+1} - t_n) + \frac{1}{2}y''(\xi)(t_{n+1} - t_n)^2$$

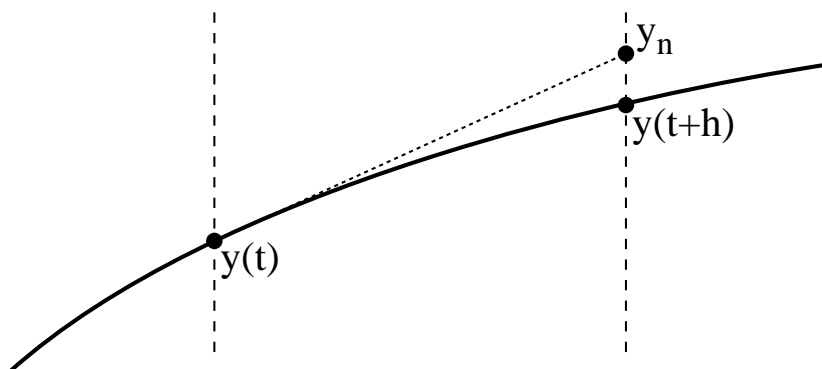


Figure 1: Forward Euler; advance along the line with slope equal to the derivate at the current point.

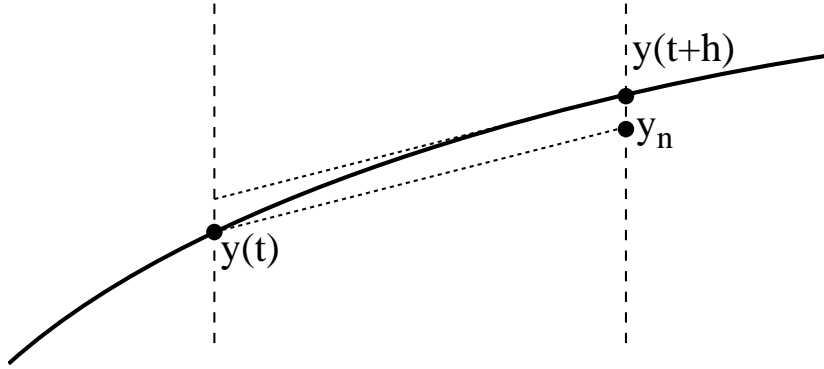


Figure 2: Backward Euler; advance along the line with slope equal to the derivate at the next point.

$$\begin{aligned}
 y(t_{n+1}) &= y(t_n) + hf(y_n, t_n) + \frac{1}{2}y''(\xi)h^2 \\
 y(t_{n+1}) &= y(t_n) + hf(y_n, t_n) + O(h^2)
 \end{aligned}$$

Thus, we see that each iteration of the scheme is second order accurate. Iterated over n steps, the accuracy is $O(h)$, or first order.

2.2 Stability

Consider $y' = \lambda y$, where $\lambda < 0$. Then, $y = e^{\lambda t} \rightarrow 0$ as $t \rightarrow \infty$. For a method to be stable it must be the case that $y_n \rightarrow 0$.

$$y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n = (1 + h\lambda)^{n+1}y_0$$

This goes to zero when $|1 + h\lambda| < 1$. Assuming λ is real, then forward Euler is stable when $h < -\lambda^{-1}$.

3 Backwards Euler

A slight modification on the forward Euler is the Backward Euler, whose update rule is:

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1})$$

The motivation behind the Backward Euler is essentially the same as the forward Euler, except the slope of the line is estimated at the other end of the update interval, as illustrated in Figure 2. This technique is not as straightforward as the Forward Euler, since it is an implicit formula. The update rule is actually an equation that must be solved at each timestep. This increases the cost of each update.

3.1 Derivation

$$\begin{aligned}
 y(t_n) &= y(t_{n+1}) + y'(t_{n+1})(t_n - t_{n+1}) + \frac{1}{2}y''(\xi)(t_n - t_{n+1})^2 \\
 y(t_{n+1}) &= y(t_n) - y'(t_{n+1})(-h) - \frac{1}{2}y''(\xi)(-h)^2 \\
 y(t_{n+1}) &= y(t_n) + hf(y_{n+1}, t_{n+1}) - \frac{1}{2}y''(\xi)h^2 \\
 y(t_{n+1}) &= y(t_n) + hf(y_{n+1}, t_{n+1}) + O(h^2)
 \end{aligned}$$

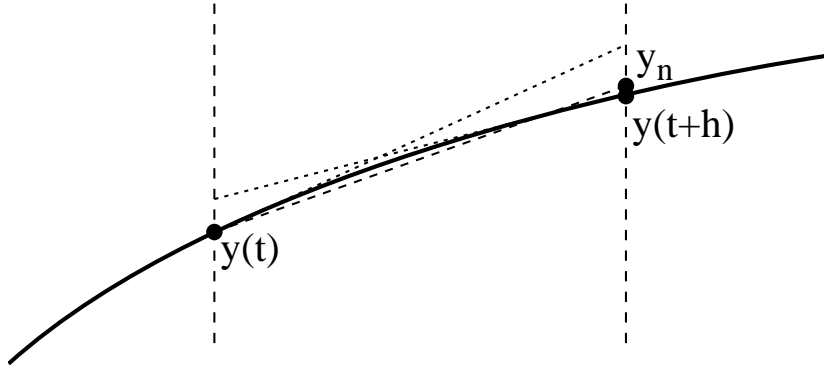


Figure 3: Trapezoid scheme; advance along the line with slope equal to the average of the derivatives at the current and next points.

Again, we see that each iteration of the scheme is second order accurate. Iterated over n steps, the accuracy is $O(h)$, or first order.

3.2 Stability

$$\begin{aligned}
 y_{n+1} &= y_n + h\lambda y_{n+1} \\
 y_{n+1} - h\lambda y_{n+1} &= y_n \\
 y_{n+1} &= \frac{y_n}{1 - h\lambda} \\
 y_{n+1} &= \frac{y_0}{(1 - h\lambda)^{n+1}}
 \end{aligned}$$

This goes to zero when $|1 - h\lambda| > 1$. Assuming λ is real and negative, then backward Euler is always stable. It is the extra stability that makes backward Euler attractive, even though it is more expensive to compute due to the need to solve an equation at each timestep. The stability means that larger timesteps can be used than are possible with the forward Euler update.

4 Trapezoidal

The trapezoid rule is more complicated than the above and essentially combines their ideas. It takes an average of the slopes at each side and uses that as the slope of a line to follow. The update formula for the trapezoid scheme is:

$$y_{n+1} = y_n + \frac{h}{2}(f(y_n, t_n) + f(y_{n+1}, t_{n+1})).$$

The trapezoid scheme is illustrated in Figure 3. Like the backwards Euler, this formula is also implicit and requires an equation to be solved at each timestep. The extra expense of this technique is easily paid for by its higher order of convergence.

4.1 Derivation

$$y(t_{n+1}) = y(t_n) + y'(t_n)(t_{n+1} - t_n) + \frac{1}{2}y''(t_n)(t_{n+1} - t_n)^2 + \frac{1}{6}y'''(\xi)(t_{n+1} - t_n)^3$$

$$\begin{aligned}
y(t_{n+1}) &= y(t_n) + hy'(t_n) + \frac{1}{2}h^2y''(t_n) + \frac{1}{6}y'''(\xi)h^3 \\
y(t_n) &= y(t_{n+1}) + y'(t_{n+1})(t_n - t_{n+1}) + \frac{1}{2}y''(t_{n+1})(t_n - t_{n+1})^2 + \frac{1}{6}y'''(\xi)(t_n - t_{n+1})^3 \\
y(t_{n+1}) &= y(t_n) + hy'(t_{n+1}) - \frac{1}{2}h^2y''(t_{n+1}) + \frac{1}{6}y'''(\xi)h^3 \\
2y(t_{n+1}) &= 2y(t_n) + h(y'(t_n) + y'(t_{n+1})) + \frac{1}{2}h^2(y''(t_n) - y''(t_{n+1})) + O(h^3) \\
y''(t_{n+1}) &= y''(t_n) + y'''(\xi)(t_{n+1} - t_n) \\
2y(t_{n+1}) &= 2y(t_n) + h(y'(t_n) + y'(t_{n+1})) - \frac{1}{2}h^3y'''(\xi) + O(h^3) \\
y(t_{n+1}) &= y(t_n) + \frac{1}{2}h(f(y_n, t_n) + f(y_{n+1}, t_{n+1})) + O(h^3)
\end{aligned}$$

This time, we see that each iteration of the scheme is third order accurate. Iterated over n steps, the accuracy is $O(h^2)$, or first order.

4.2 Stability

$$\begin{aligned}
y_{n+1} &= y_n + h(\lambda y_{n+1} + \lambda y_{n+1}) \\
y_{n+1} - h\lambda y_{n+1} &= y_n + h\lambda y_n \\
y_{n+1} &= \left(\frac{1+h\lambda}{1-h\lambda}\right) y_n \\
y_{n+1} &= \left(\frac{1+h\lambda}{1-h\lambda}\right)^{n+1} y_0 \\
\left|\frac{1+h\lambda}{1-h\lambda}\right| &< 1 \\
\frac{1+h\lambda}{1-h\lambda} &< 1 \\
1+h\lambda &< 1-h\lambda \\
2h\lambda &< 0 \\
\lambda &< 0
\end{aligned}$$

Assuming λ is real and negative, then the trapezoid scheme is always stable. This feature makes the update attractive, since even larger timesteps may be taken than the backwards Euler, due to its higher order of convergence and its stability.

