TMA4320 Cheat Sheet

v1.1 (TeX source @ GitHub.com/JakobGM/)

Solving equations

Definition. The function $f(x)$ has a **root** at $x = r$ if $f(r) = 0$.

The Bisection Method

Given initial interval $[a, b]$ such that $f(a)f(b) < 0$
while $(b - a)/2 > \text{TOL}$
  $c = (a + b)/2$
  if $f(c) = 0$, **stop**, **end**
  if $f(a)f(c) < 0$
    $b = c$
  else
    $a = c$
  end
end
The final interval $[a, b]$ contains a root.
The approximate root is $(a + b)/2$.

The bisection method's efficiency:

$$\text{Solution error} = |x_c - r| < \frac{b-a}{2^{n+1}}$$

$$\text{Function evaluations} = n + 2$$

Fixed point iteration

$$x_0 = \text{initial guess}$$

$$x_{i+1} = g(x_i) \text{ for } i = 0, 1, 2, \ldots$$

```
% Function handle g
% Starting guess x0
% Number of iteration steps k
function xc = fpi(g, x0, k)
x(1) = x0;

for i = 1:k
    x(i+1) = g(x(i));
end

xc = x(k+1);
```

Theorem. Assume that g is continously differentiable, that $g(r) = r$, and that $S = |g'(r)| < 1$. The Fixed-Point Iteration converges linearly with the rate S to the fixed point r for initial guesses sufficiently close to r.

Newton's method

$$x_0 = \text{initial guess}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \text{ for } i = 0, 1, 2, \ldots$$

Theorem. Let f be twice continuously differentiable and $f(r) = 0$. If $f'(r) \neq 0$, then Newton's method is locally and quadratically convergent to r. The error e_i at step i satisfies **quadratic convergence**

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} = M,$$

where

$$M = \frac{f''(r)}{2f'(r)}.$$

Theorem. Assume that the $(m+1)$-times continuously differentiable function f on $[a, b]$ has a multiplicity m at root r. Then Newton's Method is locally convergent to r, and the error e_i at step i satisfies

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i} = S,$$

where $S = (m-1)/m$.

Theorem. If f is $(m+1)$-times continuously differentiable on $[a, b]$, which contains a root r of multiplicity $m > 1$, the the **Modified Newton's Method**

$$x_{i+1} = x_i - \frac{mf(x_i)}{f'(x_i)}$$

converges locally and quadratically to r.

Interpolation

Lagrange interpolation

The *unique* degree $n - 1$ polynomial that interpolates the n datapoints $(x_1, y_1), \ldots, (x_n, y_n)$ is given by

$$P_{n-1}(x) = y_1 L_1(x) + \ldots + y_n L_n(x)$$

where L_k is given by

$$L_k(x) = \frac{(x - x_1)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_n)}{(x_k - x_1)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_n)}$$

Netwon's divided differences

Definition. Denote by $f[x_1 \ldots x_n]$ the coefficient of the x^{n-1} term in the (unique) polynomial that interpolates $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$.

Given $x = [x_1, \ldots, x_n]$, $y = [y_1, \ldots, y_n]$

for $j = 1, \ldots, n$
  $f[x_j] = y_j$
end

for $i = 2, \ldots, n$
  for $j = 1, \ldots, n + 1 - i$
    $f[x_j \ldots x_{j+i-1}] = (f[x_{j+1} \ldots x_{j+i-1}] - f[x_j \ldots x_{j+i-2}])/(x_{j+i-1} - x_j)$
  end
end
The interpolating polynomial is

$$P(x) = \sum_{i=1}^{n} f[x_1 \ldots x_i](x - x_1)\cdots(x - x_{i-1})$$

A recursive table in the form

$$\begin{array}{c|cccc}
x_1 & f[x_1] & & \\
 & & f[x_1\ x_2] & \\
x_2 & f[x_2] & & f[x_1\ x_2\ x_3] \\
 & & f[x_2\ x_3] & \\
x_3 & f[x_3] & &
\end{array}$$

can be made, and the top row gives the coefficients of the Newton's divided difference polynomial.

Interpolation error

Theorem. Assume that $P(x)$ is the (degree $n - 1$ or less) interpolating polynomial fitting the n points $(x_1, y_1), \ldots, (x_n, y_n)$. The interpolation error is

$$f(x) - P(x) = \frac{(x - x_1)(x - x_2)\ldots(x - x_n)}{n!} f^{(n)}(c),$$

where $c \in [\min(x_1, \ldots, x_n), \max(x_1, \ldots, x_n)]$.

Runge's phenomenon

Runge's phenomenon is the consequence of the magnitude of the derivatives of the interpolation function grows quickly when n increases. This causes a "wiggle" effect at the ends of the interval and is solved by redistributing the interpolation nodes towards the ends. Speaking of which...

Chebyshev Interpolation Nodes

Theorem. On the interval $[a,b]$,

$$x_i = \frac{b+a}{2} + \frac{b-a}{2}\cos\frac{(2i-1)\pi}{2n}$$

for $i = 1,...,n$. The inequality

$$|(x-x_1)...(x-x_n)| \leq \frac{(\frac{b-a}{2})^n}{2^{n-1}}$$

holds on $[a,b]$. The use of these nodes will minimize the interpolation error.

Numerical quadratures

Methods for integrating $f(x)$ on the interval $[a,b]$, using m points. The used variable c is always contained in this interval.

Composite Trapezoid Rule

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{2}(y_0 + y_m + 2\sum_{i=1}^{m-1} y_i) - \frac{(b-a)h^2}{12}f''(c),$$

where $h = (b-a)/m$.

Composite Midpoint Rule

Functions with removable singularities at an interval endpoint can be handled with

$$\int_a^b f(x)\,\mathrm{d}x = h\sum_{i=1}^{m} f(w_i) + \frac{(b-a)h^2}{24}f''(c),$$

where $h = (b-a)/m$. The w_i are the midpoints of m equal subintervals of $[a,b]$.

Higher order quadratures

To find the Newton-Cotes quadrature of the nth degree, use the Lagrange polynomial of the nth degree with its interpolation error term given above

$$\int_{x_0}^{x_n} f(x)\,\mathrm{d}x = \int_{x_0}^{x_n} P_n(x) + E_n(x)\,\mathrm{d}x,$$

where

$$\int_{x_0}^{x_n} P_n = \sum_{i=0}^{n} f(x_i)\int_{x_0}^{x_n} L_k(x)\,\mathrm{d}x.$$

The degree of precision is n (for n odd) and $n+1$ (for n even), with $n+1$ function evaluations.

Gaussian quadrature

Definition. The set of nonzero functions $\{p_0,...,p_n\}$ on the interval $[a,b]$ is **orthogonal** on $[a,b]$ if

$$\int_a^b p_j(x)p_k(x)dx = \begin{cases} 0 & j \neq k \\ \neq 0 & j = k \end{cases}$$

Theorem. These orthogonal polynomials, where deg $p_i = i$, form a basis for the vector space of degree at most n polynomials on $[a,b]$. p_i then has i distinct roots in the interval (a,b).

The set of **Legendre polynomials**

$$p_i(x) = \frac{1}{2^i i!}\frac{\mathrm{d}^i}{\mathrm{d}x}[(x^2-1)^i], \text{ for } 0 \leq i \leq n$$

is orthogonal on $[-1,1]$.

Gaussian quadrature of the nth degree is derived from integrating an interpolating polynomial of $f(x)$ whose nodes are the Legendre roots of p_n.

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} c_i f(x_i),$$

where

$$c_i = \int_{-1}^{1} L_i(x)\,\mathrm{d}x, \; i = 1,...,n.$$

For a general interval $[a,b]$, use the substitution $t = (2x-a-b)/(b-a)$ to translate back to $[-1,1]$. Gaussian quadrature, using a degree n Legendre polynomial, has a degree of precision of $2n-1$.

Adaptive quadrature

Denote the error estimation of the non-composite quadrature method $S_{[a,b]}$ on the interval $[a,b]$ as $E_S(a,b)$. For the trapezoid rule for instance, we have $E_{\text{trap}}(a,b) = -h^3 f''(c_0)/12$. The factor of error estimation reduction, r_s, when halving the interval length, $h \to h/2$, is equal to

$$r_S = \left|\frac{E_S(a,b) - (E_S(a,c) + E_S(c,b))}{E_S(a,c) + E_S(c,b)}\right|,$$

where $c = (a+b)/2$. For the trapezoid rule, r_{trap} is equal to 3. When calculating $S_{[a,b]}$, the error bound can be compared with the specified tolerance, TOL, by evaluating

$$\left|E_S(a,b) - (E_S(a,c) + E_S(c,b))\right| < r_S \cdot \frac{\text{TOL}}{2^n},$$

where n is equal to how many times the original interval has been halved. An example for the trapezoid rule is given:

To approximate $\int_a^b f(x)\,dx$ within tolerance TOL:

$c = \dfrac{a+b}{2}$

$S_{[a,b]} = (b-a)\dfrac{f(a)+f(b)}{2}$

if $|S_{[a,b]} - S_{[a,c]} - S_{[c,b]}| < 3 \cdot \text{TOL} \cdot \left(\dfrac{b-a}{b_{\text{orig}} - a_{\text{orig}}}\right)$

    accept $S_{[a,c]} + S_{[c,b]}$ as approximation over $[a,b]$

else

    repeat above recursively for $[a,c]$ and $[c,b]$

end

Error estimation

Integrate the interpolation error (see "Interpolation Error") or perform a Taylor series expansion and integrate the error term (see "Taylor Method of order k").

ODEs

Solving the initial value problem

$$\begin{cases} y' = f(t,y) \\ y(a) = y_a \\ t \in [a,b] \end{cases}$$

with...

Euler's method

$$\begin{aligned} w_0 &= y_0 \\ w_{i+1} &= w_i + hf(t_i, w_i) \end{aligned}$$

Backwards Euler Method

Use this method when the differential equation is **stiff**, i.e. attracting solutions are surrounded with fast-changing nearby solutions, i.e. when the linear part of y on the r.h.s. is large and negative.

$$\begin{aligned} w_0 &= y_0 \\ w_{i+1} &= w_i + hf(t_{i+1}, w_{i+1}) \end{aligned}$$

Solving this implicit equation for w_{i+1} might require the iterative use of Newton's method.

Explicit Trapezoid Method

$$
\begin{aligned}
w_0 &= y_0 \\
w_{i+1} &= w_i + \frac{h}{2}(f(t_i, w_i) + f(t_i + h, w_i + hf(t_i, w_i)))
\end{aligned}
$$

Local and global error

Definition. A function $f(t, y)$ is **Lipschitz continuous** in the variable y on the rectangle $S = [a, b] \times [\alpha, \beta]$ if there exists a constant L (called the **Lipschitz constant**) satisfying

$$\left|f(t, y_1) - f(t, y_2)\right| \leq L|y_1 - y_2|$$

for each (t, y_1),(t, y_2) in S.

Definition. The **global truncation error** is defined as $g_i = |w_i - y_i|$, and the **local truncation error** is defined as $e_{i+1} = \left|w_{i+1} - z(t_{i+1})\right|$, where z is the correct solution of the one-step IVT with $y_0 = w_i$.

Theorem. If $f(t, y)$ has a Lipschitz constant L, and the ODE solver has a local truncation error $e_i \leq Ch^{k+1}$, then the solver (which is of order k) has a global truncation error

$$g_i = |w_i - y_i| \leq \frac{Ch^k}{L}(e^{L(t_i - a)} - 1).$$

Taylor Method of order k

$$
\begin{aligned}
w_0 =& \ y_0 \\
w_{i+1} =& \ w_i + hf(t_i, w_i) + \frac{h^2}{2} f''(t_i, w_i) \\
&+ ... + \frac{h^k}{k!} f^{(k-1)}(t_i, w_i)
\end{aligned}
$$

with the corresponding error term

$$y_{i+1} - w_{i+1} = \frac{h^{k+1}}{(k+1)!} y^{(k+1)}(c) = \mathcal{O}(h^{k+1}),$$

where $c \in [t, t + h]$.

Adaptive methods

Compare e_i or $e_i/\mathtt{max}(|w_i|, \theta)$ with the error tolerance, and change h_i as needed. If $e_i \approx ch_i^{p+1}$, the relative tolerance $\mathtt{TOL}$ is satisfied when $\mathtt{TOL} \leq ch^{p+1}/|w_i|$. Solving for h gives the new step size

$$h_{i+1} = 0.8 \left(\frac{\mathtt{TOL} \cdot |w_i|}{e_i} \right)^{\frac{1}{p+1}} h_i,$$

with a safety factor of 0.8.

Embedded pairs

The error in going from t_i to t_{i+1} can be estimated as $e_{i+1} \approx |z_{i+1} - w_{i+1}|$, where z is a higher order estimate. This is often done with an **embedded Runge-Kutta pair** that shares much of the needed computations. An example is the order 2/order 3 embedded pair:

$$
\begin{aligned}
w_{i+1} &= w_i + h\frac{s_1 + s_2}{2} \\
z_{i+1} &= w_i + h\frac{s_1 + 4s_3 + s_2}{6}
\end{aligned}
$$

where

$$
\begin{aligned}
s_1 &= f(t_i, w_i) \\
s_2 &= f(t_i + h, w_i + hs_1) \\
s_3 &= f(t_i + \frac{h}{2}, w_i + \frac{h}{2}\frac{s_1 + s_2}{2})
\end{aligned}
$$

with an error estimation of

$$e_{i+1} \approx |w_{i+1} - z_{i+1}| = \left| h\frac{s_1 - 2s_3 + s_2}{3} \right|.$$

It's of course better to use z_{i+1} to advance the step (**local extrapolation**).

Higher order equations

A single ordinary differential equation of nth order,

$$y^{(n)} = f(t, y, y', y'', ..., y^{(n-1)}),$$

can be converted to a solvable system of n *first order* differential equations by defining new variables $y_i = y^{(i-1)}$ for $i = 1, ..., n-1$, and writing the original equation as $y_n' = f(t, y_1, y_2, ..., y_n)$. This gives the following system of first-order equations

$$
\begin{aligned}
y_1' &= y_2 \\
y_2' &= y_3 \\
y_3' &= y_4 \\
&\vdots \\
y_{n-1}' &= y_n \\
y_n' &= f(t, y_1, ..., y_n)
\end{aligned}
$$

which can be solved by the methods mentioned earlier in this section.

DFT/FFT

Definition. The **Discrete Fourier Transform** of $x = [x_0, ..., x_{n-1}]^T$ is the n-dimensional vector $y = [y_0, ..., y_{n-1}]^T$, where $w = e^{-i2\pi/n}$ and

$$y_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j w^{jk}.$$

Or in matrix terms

$$
\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} a_0 + ib_0 \\ a_0 + ib_1 \\ a_0 + ib_2 \\ \vdots \\ a_{n-1} + ib_{n-1} \end{bmatrix} = F_n \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix},
$$

where the **Fourier matrix**, F_n, is equal to

$$
F_n = \frac{1}{\sqrt{n}} \begin{bmatrix}
w^0 & w^0 & w^0 & \dots & w^0 \\
w^0 & w^1 & w^2 & \dots & w^{n-1} \\
w^0 & w^2 & w^4 & \dots & w^{2(n-1)} \\
w^0 & w^3 & w^6 & \dots & w^{3(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
w^0 & w^{n-1} & w^{2(n-1)} & \dots & w^{(n-1)^2}
\end{bmatrix}
$$

The **inverse Discrete Fourier Transform** is then given by $x = F_n^{-1} y$ or

$$x_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} y_j w^{-jk}.$$

Useful properties of the DFT

- The inverse of the Fourier matrix is the matrix consisting of the complex conjugates of the entries of F_n: $F_n^{-1} = \overline{F}_n$.

- The Fourier matrix is **unitary**, that is $\overline{F}_n^T F_n = I$.
- The magnitude of a complex vector is: $||\vec{v}|| = \sqrt{\overline{\vec{v}}^T \vec{v}}$.
- There is no change in magnitude after a unitary matrix multiplication: $||Fv||^2 = \overline{v}^T \overline{F}^T F v = \overline{v}^T v = ||v||^2$.
- If $\vec{x}$ is real, then y_0 is real, and $y_{n-k} = \overline{y_k}$.
- $\vec{x_1} \cdot \vec{x_2} = \vec{x_2}^T \vec{x_1}$ and $[F_n \vec{x}]^T = \vec{x}^T F_n^{-1}$.

The Fast Fourier Transform

The FFT uses the following property in order to split DFT(N) into two DFT($N/2$)s plus $2N-1$ extra operations

$$\sum_{n=0}^{N-1} x_n e^{-2\pi i n k/N}$$

$$= \sum_{n=0}^{N/2-1} x_{2n} e^{-2\pi i (2n)k/N} + \sum_{n=0}^{N/2-1} x_{2n+1} e^{-2\pi i (2n+1)k/N}$$

$$= \sum_{n=0}^{N/2-1} x_n^{\text{even}} e^{-2\pi i n k/(N/2)} + e^{-2\pi i k/N} \sum_{n=0}^{N/2-1} x_n^{\text{odd}} e^{-2\pi i n k/(N/2)}$$

Trigonometric interpolation

Given the interval $[c, d]$ and positive integer n, let $t_j = c + j(d-c)/n$ for $j = 0, ..., n-1$, and let $\vec{x} = (x_0, ..., x_{n-1})$ denote a vector of n numbers. Define $\vec{a} + \vec{b}i = F_n \vec{x}$. Then the complex function

$$Q(t) = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} (a_k + i b_k) e^{i2\pi k(t-c)/(d-c)}$$

satisfies $Q(t_j) = x_j$ for $j = 0, ..., n-1$. Furthermore, if the x_j are real, the function

$$P(t) = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \left(a_k \cos \frac{2\pi k(t-c)}{d-c} - b_k \sin \frac{2\pi k(t-c)}{d-c} \right)$$

satisfies $P(t_j) = x_j$ for $j = 0, ..., n-1$, assuming n is even. Using the cosine and sine addition formulas together with the fact that $y_{n-k} = \overline{y_k}$, $P(t)$ can be simplified to

$$P_n(t) = \frac{a_0}{\sqrt{n}} + \frac{2}{\sqrt{n}} \sum_{k=1}^{n/2-1} \left(a_k \cos \frac{2\pi k(t-c)}{d-c} - b_k \sin \frac{2\pi k(t-c)}{d-c} \right) + \frac{a_{n/2}}{\sqrt{n}} \cos \frac{n\pi(t-c)}{d-c}$$

Fourier filtering/compression relevant to the project

- `MATLAB` uses a non-unitary normalization for its Fourier transformation, such that $F_n x$ is computed by `fft(x)/sqrt(n)`, and $F_n^{-1} y$ by `ifft(y)*sqrt(n)`.
- Given n data points, the best least squares trigonometric function with $m < n$ terms can be found by interpolating with n terms, and then only keep the first m terms (dropping the higher frequencies, called a **low pass filter**).
- A **high pass filter** can be made by dropping the lower frequency components.