COMPUTATIONAL METHODS: ABRIDGED

HENRY O'HAGAN

Contents

1. Basic ODEs, Error types, Differentiation and Sundries

1.1. **Errors.** Types of errors: (Qualitatively)

- *Truncation error* – Where the value of a function is only calculated to an approximate value, which can be compared to truncating the taylor series: e.g. $\sin(x) \approx x - \frac{x^3}{3!} =: p_3(x) \implies \epsilon(x) \equiv sin(x) - p_3(x) \sim (O)(x^5)$
- *Round-off* – From the data type. Single precision: 1-bit for sign, 23 bits for mantissa, 8-bits for exponent, $\sim 10^{\pm 38}$ Double precision:1-bit for sign, 52 bits for mantissa, 11-bits for exponent (offset), $10^{\pm 308}$.
- *Initial conditions* – Self explanatory (chaotic systems are sensitive to this)
- *Propagation* – Self explanatory, error for each step in the program: if it increases with each step the solution is said to be *unstable*, otherwise it is said to be *stable*.

1.2. **Numerical Differentiation.**

1.2.1. FDS.

$$\boxed{\bar{y}'_f(x) \equiv \frac{y(x+h) - y(x)}{h}}$$

Error $\sim \mathcal{O}(h)$.

1.2.2. BDS.

$$\boxed{\bar{y}'_b(x) \equiv \frac{y(x) - y(x-h)}{h}}$$

Error $\sim \mathcal{O}(h)$.

1.2.3. CDS.

$$\boxed{\bar{y}'_c(x) \equiv \frac{y(x+h) - y(x-h)}{2h}}$$

Error $\sim \mathcal{O}(h^2)$.

1.2.4. 2nd order.

$$\boxed{\bar{y}''(x) \equiv \frac{y(x+h) + y(x-h) - 2y(x)}{h^2}}$$

Error $\sim \mathcal{O}(h^2)$. Derived from 4^{th} order Taylor series of $y(x+h)$ and $y(x-h)$. In general, finite difference approximations of $\bar{y}^{(n)}$ requires $n+1$ points going further away from x.

1.3. **ODEs.**

1.3.1. Euler Method.

Basic example, first order. ODE of the form:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y,t)$$

Then

$$\boxed{y_{n+1} = y_n + f(t_n, y_n)h}$$

where $t_n := t_0 + (n-1)h$.

Higher order, Inhomogeneous.

$$\frac{\mathrm{d}^n y}{\mathrm{d}t^n} + a_{n-1}\frac{\mathrm{d}^{n-1}y}{\mathrm{d}t^{n-1}} + \cdots a_0 y + f(y, y', y'', \ldots y^{(n-1)}, t) = 0$$

Where the function f comes from the inhomogeniety, Introduce new variables:

$$z_0 := y \quad z_1 := y' \quad \cdots z_{n-1} := y^{(n-1)} \quad y^{(n)} = -(a_0 z_0 + \cdots a_{n-1} z_{n-1} + f(\ldots))$$

Then we have coupled first order ODEs, which can be written in matrix form:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & \cdots & -a_{n-2} & -a_{n-1} \end{pmatrix}\begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ f(\ldots) \end{pmatrix}$$

Summarised using index notation as

$$\boxed{\frac{\mathrm{d}z_i}{\mathrm{d}t} = L_{ij}z_j - \delta_{i(n-1)}f(z_k, t)}$$

The euler method is then implemented thusly:

$$[z_i]_{m+1} = [z_i]_m + h\frac{\mathrm{d}[z_i]_m}{\mathrm{d}t}$$

$$\implies \boxed{[z_i]_{m+1} = [\delta_{ij} + hL_{ij}][z_j]_m - \delta_{i(n-1)}f(z_k, t)h}$$

$\delta_{ij} + hL_{ij}$ is known as the "update matrix". If the function is non-linear, it can be linearised by taylor expanding to some desired order, then can be made into a linear operator.

1.4. **Natural Units.** Example:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \alpha y$$

$[\alpha] = s^{-1}$ define new variable, $\widetilde{t} := \alpha t$

$$\implies \frac{\mathrm{d}y}{\mathrm{d}\widetilde{t}} = y$$

System is now in dimensionless units.
Advantages:

- Step size: h is now small for $h < 1$. If we still had dimensions, then it would be harder to determine what is meant by "small".
- Range of integration: The characteristic timescale in dimensionless units is $\widetilde{t} \sim 1$ so we know integrating from $\widetilde{t} = 0$ to $\widetilde{\sim}$ a few will cover the entire dynamic range of interest.

N.B. Remember to put units back when we present plots.

1.5. **Finding Roots.**

1.5.1. Bisection. use

$$\boxed{x_m = \frac{x_l + x_r}{2}}$$

s.t. $sgn(f(x_l)) = -sgn(f(x_r))$ i.e. such that they have opposite sign. Continue to iterate depending on the sign of x_m .
This converges linearly with error halving each iteration. $\epsilon_{i+1} = \epsilon_i/2$

1.5.2. Newton's Method. (You've probably done this before)

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

This can easily fail if start points are near maxima or minima. This converges quadratically, $\epsilon_{i+1} = \text{const} \times \epsilon_i^2$.

1.5.3. Secant Method. Using Newton's method with an approximate derivative:

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

$$\implies \boxed{x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) + f(x_{i-1})}}$$

1.6. **Interpolation.** Finding the value of a function between two known points.

1.6.1. Linear Interpolation. Using straight line approximation where x is in between:

$$\boxed{f(x) = \frac{(x_{x_I+1} - x)f_i + (x - x_i)f_{i+1}}{x_{i+1} - x_i}}$$

1.6.2. Lagrange Polynomials. given $n + 1$ points (x_i, f_i), then it can be approximated by

$$\boxed{P_n(x) = \sum_{i=0}^{n} \left(\prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \right) f_i}$$

1.6.3. Bi-Linear Interpolation. Same as Linear but for higher dimensions, doing it one dimension at a time. (Look it up if you must)

2. The Quality of a Computational Solution

2.1. **Consistency.** This is a check that the finite difference method reduces to the correct DE in the limit that the step size approaches 0. Also reveals the differential equation that the finite difference method is *actually* solving. Consistency analysis also tells us the order of the method. This is useful if one is given a finite difference equation but not told what the order of accuracy is.

This is all done by expanding the finite difference equation about the base point η_n.
Consistency analysis done on Euler Method:

$$y_{n+1} = y_n + f(\eta_n, y_n)h$$

$$y_n + y_n' h + \frac{y_n''}{2}h^2 + \ldots = y_n + f(\eta_n, y_n)h \qquad (y' := \mathrm{d}y/\mathrm{d}\eta)$$

$$\implies y_n' = f(\eta_n, y_n) - \frac{y_n''}{2}h - \frac{y_n'''}{3!}h^2 - \ldots$$

Making this continuous: $\eta_n \to \eta$:

Modified DE (MDE)
$$\boxed{y' = f(\eta, y) - \frac{y''}{2}h - \frac{y'''}{3!}h^2 - \ldots}$$

The discrete points lie on the continuous curve that solves this (given the same initial conditions). let $h \to 0$

$$\text{MDE} \implies y' = f(\eta, y)$$

As expected. Thus 3.1. is consistent with the original. The lowest order in h is in this case $-y''h/2 \sim \mathcal{O}(h)$ the order of accuracy of the method. This is the same as the order of the global error.

2.2. **Accuracy.** Takes local error (truncation) and uses it to estimate global error.

$$\widetilde{y}_{n+1} = y_n + f_n h \equiv y_{n+1} + \mathcal{O}(h^2)$$

Integrating from η_0 to η_{end} would require $\mathcal{O}(1/h)$ steps. Assuming local error simply adds up:

$$\text{global error} \approx \text{local error} \times \text{number of steps}$$

Then $\epsilon_{\text{end}} \approx \mathcal{O}(h^2) \times \mathcal{O}(1/h) \sim \mathcal{O}(h)$.

A method is called n^{th} -order if its local error is $\mathcal{O}(h^{n+1})$, it's global accuracy $\mathcal{O}(h^n)$.

The round-off error gives a limit on how accurate any method can be. An error $\mathcal{O}(\mu)$ is added every step in integration, then the global error after $\mathcal{O}(1/h)$ steps will be $\epsilon_{\text{end}} \sim \frac{\mu}{h} + h$. Differentiate wrt h and set to 0 gives a $h \sim \mu^{1/2}$. For double precision $h \sim 10^{-8}$ which gives $\epsilon_{\text{end}} \sim 10^{-8}$.

2.3. **Stability.** Describes how error propagates.

2.3.1. uncoupled systems.

$$\widetilde{y}_n = y_n + \epsilon_n$$

Where y_n is the true solution of the ODE at $\eta = \eta_n$ and $\widetilde{y}_n$ the numerical approximation. Amplification factor:

$$\boxed{g := \left|\frac{\epsilon_{n+1}}{\epsilon_n}\right| \leq 1}$$

or else the error 'blows up'.

Take the Euler method as an example:

$$y_{n+1} + \epsilon_{n+1} = y_n + \epsilon_n + f(\eta_n, y_n + \epsilon_n)h$$

Stability analysis can only be done if f is linear. If f is a non-linear function, we can still do the analysis by assuming $|\epsilon_{n+1}| \ll |y_n|$ and linearising $f(\eta_n, y_n + \epsilon_n)$ by expanding it around (η_n, y_n) in y.

$$y_{n+1} + \epsilon_{n+1} = y_n + \epsilon_n + \left[f(\eta_n, y_n) + \left.\frac{\partial f}{\partial y}\right|_n \epsilon_n + \mathcal{O}(\epsilon_n^2)\right] h$$

$$= y_n + f(\eta_n, y_n)h + \epsilon_n \left[1 + \left.\frac{\partial f}{\partial y}\right|_n h\right] + \mathcal{O}(\epsilon_n^2)$$

Since $y_{n+1} = y_n + f(\eta_n, y_n)h + \mathcal{O}(h^2)$

$$\boxed{\epsilon_{n+1} \approx \epsilon_n \left(1 + \left.\frac{\partial f}{\partial y}\right|_n h\right)}$$

From the previous condition:

$$g \leq 1 \implies -2 \leq \frac{\partial f}{\partial y} h \leq 0$$

Assuming $h > 0$ then stability conditions are:

$$h \leq \frac{2}{|\partial_y f|}, \quad \partial_y f < 0$$

"*Conditionally stable*" for $\partial_y f < 0$, " *Unconditionally unstable* " for $\partial_y f > 0$ (no step size works). Linear implies conditions only depend on independent variable, non-linear implies conditions also depend on y. (Stability doesn't tell us about global error.)

2.3.2. Coupled Linear Systems. Finite difference method written as:

$$[\widetilde{y}_i]_{n+1} = T_{ij}[\widetilde{y}_j]_n + q_i(\eta)$$

q_i (for inhomogeneous does not influence the numerical stability. Then inserting errors:

$$[\widetilde{y}_i]_{n+1} + [\epsilon_i]_{n+1} = T_{ij}([\widetilde{y}_j]_n + [\epsilon_j]_n)$$

Taylor expand as before:

$$[\widetilde{y}_i]_{n+1} = [\widetilde{y}_i]_n + [f_i]_n h + \mathcal{O}(h^2) \approx T_{ij}[y_j]_n$$

inserting into the previous equation, ignoring everything of order 2 yields:

$$\boxed{[\epsilon_i]_{n+1} = T_{ij}[\epsilon_j]_n}$$

This (non-trivially) implies that

$$|\lambda_i| \leq 1 \forall i$$

Where the lambdas are the eigenvalues of the update matrix.

Derivation:

$$(T_{ij} - \lambda_i \delta_{ij}) v_j = 0$$

Then eigendecomposition gives $T_{ij} = R_{ik} D_{kl} R^{-1}_{lj}$ where R is a matrix whose columns are eigenvectors of T_{ij}. Subbing in the diagonalised form into the error propagation formula gives:

$$[\zeta_i]_{n+1} = D_{ij}[\zeta_j]_n = \lambda_i [\zeta_i]_n$$

where ζ is the vector right-multiplied by the inverse of R, which rotates the coupled vectors onto an uncoupled basis. We impose

$$g_i = \left| \frac{[\zeta_i]_{n+1}}{[\zeta_i]_n} \right| = |\lambda_i| \leq 1$$

Since the coupled and uncoupled bases are related by a linear transformation, any constraint applied in one basis is equivalent to one applied in the other.

2.3.3. Non-Linear Systems. Need to perform "linearisation" again:

$$[\widetilde{y}_i]_{n+1} = [\widetilde{y}_i]_n + f_i(\eta, [\widetilde{y}_i]_n) h$$

$$[y_i]_{n+1} + [\epsilon_i]_{n+1} = [y_i]_n + [\epsilon_i]_n + f_i(\eta, [\widetilde{y}_i]_n) h$$

substituting errors in f, and taylor expand about true solution:

$$f_j(\eta, [y_i]_n + [\epsilon_i]_n) = f_j(\eta, [y_i]_n) + [\epsilon_i]_n [\frac{\partial}{\partial y_i} f_j]_n + \mathcal{O}([\epsilon_i]_n^2)$$

subbing back into previous formula and taylor expanding $[y_i]_{n+1}$ about η and cancelling things of order h^2 and ϵ^2, gives:

$$\boxed{[\epsilon_i]_{n+1} \approx [\epsilon_i]_n + h[\epsilon_j]_n \left. \frac{\partial f_i}{\partial y_j} \right|_n}$$

The new update matrix comes from comparing this with the previous equation for updating errors ($[\epsilon_i]_{n+1} = T_{ij}[\epsilon_j]_n$) :

$$\implies \boxed{T_{ij} = \delta_{ij} + h\partial_j f_i}$$

Where $\partial_j := \partial/\partial y_j$.

Then the eigendecomposition etc. can be performed on this modified form.

2.4. **Convergence.** If consistent and stable, then it converges to true solution as step size goes to 0.

2.5. **Efficiency.** Number of computations per step.

3. Methods for Integrating ODE systems

3.1. **Predictor-Corrector.** Euler method using average of gradient between 2 points:

$$y_{n+1} = y_n + h\left(\frac{f_n + f_{n+1}}{2}\right)$$

Global error:

$$y_{n+1} = y_n + f_n h + f'_n\frac{h^2}{2} + f''_n(\xi)\frac{h^3}{3!}$$

where as usual $\eta_n < \xi < \eta_n + h$ (using mean value theorem). $f_{n+1} = f_n + f'_n h + f''_n(\zeta)\frac{h^2}{2}$.

$$\implies f'_n = \frac{f_{n+1} - f_n}{h} - f''_n(\zeta)\frac{h}{2}$$

$$\implies y_{n+1} = y_n + f_n h + \frac{f_{n+1} - f_n}{2h}h^2 - \left(f''_n(\zeta)\frac{h}{2}\right)\frac{h^2}{2} + \frac{h^3}{3!}f''_n(\xi)$$

$$\implies y_{n+1} = y_n + \frac{f_{n+1} + f_n}{2}h + \mathcal{O}(h^3)$$

So the local error is 3rd order, global error is 2nd order.

For f_{n+1} , we still need $y_n + 1$ first.

(step 1 – predict) $y^*_{n+1} = y_n + f_n h$

$f^*_{n+1} = f(\eta_{n+1}, y^*_{n+1})$

(step 2 – correct) $y^{\text{new}}_{n+1} = y_n + \frac{f^*_{n+1} + f_n}{2}h$

Easily generalised to N-D. It is conditionally stable for decaying solutions but unstable for oscillatory solutions.

3.2. **Multi-step (Leapfrog).**

3.2.1. Basic method.

$$\boxed{y_{n+1} = y_{n-1} + 2f_n h}$$

Uses CDS to approximate y'. It's an explicit FDM. $\mathcal{O}(h^2)$ accurate, and requires one evaluation per step; hence more efficient than Predictor-Corrector method. Stable for oscillatory and growing solutions but unstable for decaying solutions.

3.2.2. General Multi-Step. Using m previous values . An m^{th} order multi-step method. In general an m^{th}-order multi-step method is equivalent to fitting $P_m(\eta)$, then approximating y' by P'_m . $m+1$ points are required. So values before η_0 need to be guessed, if the guess is not a good one the method will suffer from initial conditions error.

3.2.3. *Starting off.* Use something like Euler for the initial step:

$$\begin{aligned} y_1 =& y_0 + f_0 h \\ y_2 =& y_0 + 2f_1 h \\ &\vdots \\ y_{n+1} =& y_{n-1} + 2f_n h \end{aligned}$$

To improve accuracy better single step or smaller starting steps can be used.

3.2.4. *Variable Step Size.* Easy for single steps, harder for multistep (for obvious reasons).

3.3. **Runge-Kutta.** $RK-m$ method requires an m^{th} - Order Taylor expansion, and is an m^{th} - Order finite difference method. Local error $\sim \mathcal{O}(h^{m+1})$, global error $\sim \mathcal{O}(h^m)$.

3.3.1. *RK2.*

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

Different coefficients a, b, α, β gives rise to different methods.

(Single Mid-Point)	$a = 0$	$b = 1$	$\alpha = \frac{1}{2}$	$\beta = \frac{1}{2}$
(Predictor-Corrector)	$a = \frac{1}{2}$	$b = \frac{1}{2}$	$\alpha = 1$	$\beta = 1$
(Small Error RK2)	$a = \frac{1}{3}$	$b = \frac{2}{3}$	$\alpha = \frac{3}{4}$	$\beta = \frac{3}{4}$

3.3.2. *RK4.* Uses four gradients to calculate an average one.

$$\boxed{\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= hf(\eta_n, y_n) \\ k_2 &= hf(\eta_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\ k_3 &= hf(\eta_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2) \\ k_4 &= hf(\eta_n + h, y_n + k_3) \end{aligned}}$$

Although uses 4 evaluations per step, can take much larger values of h and thus is more efficient than P-C or Euler methods. RK6 is 11 evaluations. Only $m < 4$ are $\leq m$ evaluations per step. Don't need to store any previous steps.

For a coupled linear system: Define new terms:

$$\underline{\phi} := \underline{y}_{n+1} \qquad \underline{\theta} := \underline{y}_n$$

The algorithm reduces to the equation:

$$\begin{aligned}\phi_i &= [\delta_{ij} + hA_{ij} + \frac{h^2}{2}A_{ij}^2 + \frac{h^3}{6}A_{ij}^3 + \frac{h^4}{24}A_{ij}^4]\theta_j \\ &= \sum_{n=0}^{4} \frac{1}{n!}(hA_{ij})^n\theta_j\end{aligned}$$

Where A_{ij} is the matrix which represents the derivative, i.e. $\frac{\mathrm{d}}{\mathrm{d}\eta}\underline{y} = \underline{\underline{A}}\underline{y}$

3.4. **Implicit.** Explicit methods involve evaluating $f(\eta, y)$ using known values of y. Implicit methods involve evaluating $f(\eta, y)$ using y_{n+1}. E.g. Implicit Euler:

$$\boxed{y_{n+1} = y_n + f(\eta_{n+1}, y_{n+1})h}$$

Still first order. Linear $f(\eta, y) = p(\eta)y + q(\eta)$, implicit Euler can be rearranged for $y_{n+1} = g(\eta_{n+1}, y_n)$ which can easily be solved. For f non-linear, need to use bisection/Newton/secant methods.

3.4.1. *Stability.* The Implicit Euler method is *unconditionally stable.* Stability analysis gives:

$$\frac{\epsilon_{n+1}}{\epsilon_n} \approx \left(1 - \frac{\partial f}{\partial y}h\right)^{-1}$$

decay problems give $g \leq 1$ for any positive h . The solution goes to 0 as h approaches infinity for decay problems as expected.

3.4.2. *coupled ODEs.*

$$\boxed{\begin{aligned}[y_i]_{n+1} &= [y_i]_n + hL_{ij}[y_j]_n \\ [y_i]_n &= (\delta_{ij} - hL_{ij})[y_j]_{n+1} \\ [y_i]_{n+1} &= (\delta_{ij} - hL_{ij})^{-1}[y_j]_n := T_{ij}[y_j]_n\end{aligned}}$$

L_{ij} is the same matrix operator for the implicit Euler method. T_{ij} is the update matrix for the implicit Euler. Hence we require that $\det(\underline{\underline{I}} - h\underline{\underline{L}}) \neq 0$ for the implicit euler method to work. For non-linear coupled algebraic equations solving the system is either tricky or insoluble. A way out is to yet again linearise about a known point. The update matrix needs to be calculated again at each step, since the matrix obtained by linearisation depends upon $[y_i]_n$. For this reason implicit methods are generally less efficient.

4. Matrix Algebra

4.0.1. *General Considerations.* MIT OCW's Linear algebra course might be useful.

- $A_{ij}x_j = b_i$, A is an $M \times N$, (M rows and N columns) matrix, if M=N, and all column vectors (or row vectors) are linearly independent, then there exists a nontrivial solution.
- Can also write $A_{ij}X_{jk} = B_{ik}$. then this becomes N sets of N simultaneous equations in N unknowns.
- If some rows are linear combinations of other rows, then there are less equations than there are unknowns, $M < N$ and there may not be a unique solution. Most commonly there are infinitely many solutions. The column vectors are definitely linearly dependent, since there are more column vectors than the dimension of the domain (or equivalently more column vectors than the rank plus the nullity of A).
- $M > N$ implies that there are more equations than unknowns. In general then, not all equations of N-Dimensional planes intersect simultaneously, and no solutions exist for all b_i. The solution can only be solved only for $\underline{b} \in \mathrm{Col}(\underline{\underline{A}})$.
- If the equation is $A_{ij}x_j = 0$ then the solution is non-trivial iff the columns are linearly dependent, and hence $\det(\underline{\underline{A}}) \stackrel{!}{=} 0$.
- Some of these conditions aren't "strong", it's possible for example for $M > N$ with infinitely many solutions instead of no solutions. e.g. For $\underline{\underline{A}}\underline{x} = \underline{b}$, if the column vectors of A are linearly dependent and $\underline{b}$ is in the column space of A.

More "strong" conditions are the following:

- $\mathrm{Rk}(\underline{\underline{A}}) = N = M \implies$ can find unique solution for any $\underline{b}$.
- $\mathrm{Rk}(\underline{\underline{A}}) = M < N \implies \infty$ solutions for any $\underline{b}$.
- $\mathrm{Rk}(\underline{\underline{A}}) = N < M \implies$ 0 solutions if $\underline{b} \notin \mathrm{Col}(A)$, unique solution if $\underline{b} \in \mathrm{Col}(A)$.
- $\mathrm{Rk}(\underline{\underline{A}}) < M \bigwedge \mathrm{Rk}(\underline{\underline{A}}) < N \implies$ 0 solutions if $\underline{b} \notin \mathrm{Col}(A)$, infinitely many solutions if $\underline{b} \in \mathrm{Col}(A)$.

4.1. **LU Decomposition.** comes from performing gaussian elimination until the matrix becomes upper triangular.The operations performed during gaussian elimination can be represented by lower-triangular matrices (since these are row operations the lower triangular part is left-multiplied):

$$\underline{\underline{E}}_{21}\underline{\underline{A}} = \begin{pmatrix} 1 & 0 & \cdots \\ \epsilon & 1 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \underline{\underline{A}}$$

is equivalent to saying add ϵ lots of row 1 to row 2 (and ϵ is chosen such that the element in the 2,1 position of A is reduced to 0). $\underline{\underline{E}}_{ij}$ is represented as the matrix which takes an above row to eliminate the position row i column j. Gaussian elimination consists of the same kinds of row operations which can be represented by many $\underline{\underline{E}}$ matrices.

$$\underbrace{\prod_{j=1}^{n-1}\prod_{i=1}^{m-j}\underline{\underline{E}}_{(i+j)j}}_{\underline{\underline{L}}^{-1}}\underline{\underline{A}} = \underline{\underline{U}}$$

So more succinctly:

$$\underline{\underline{A}} = \underline{\underline{L}}\,\underline{\underline{U}}$$

Example calculation of $\underline{\underline{L}}$ and $\underline{\underline{L}}^{-1}$ for a 3 by 3 matrix:

$$\underline{\underline{E}}_{21} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \underline{\underline{E}}_{31} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \quad \underline{\underline{E}}_{32} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -5 & 1 \end{pmatrix}$$

$$\underline{\underline{L}}^{-1} = \underline{\underline{E}}_{32}\underline{\underline{E}}_{31}\underline{\underline{E}}_{21} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 11 & -5 & 1 \end{pmatrix}$$

The values for $\underline{\underline{E}}^{-1}_{ij}$ are simply the same but with $-\epsilon$, i.e. add $-\epsilon$ lots of column a to column b. For the complete $\underline{\underline{L}}$, it is:

$$\underline{\underline{L}} = \left(\prod_{j=1}^{n-1}\prod_{i=1}^{m-j}\underline{\underline{E}}_{(i+j)j}\right)^{-1} = \prod_{j=1}^{n-1}\prod_{i=0}^{m-(n-j+1)}\underline{\underline{E}}^{-1}_{(m-i)(n-j)}$$

Which is the inverse of each elimination matrix $\underline{\underline{E}}_{ij}$ in the reverse order as before. In the above example:

$$\underline{\underline{L}} = \underline{\underline{E}}^{-1}_{21}\underline{\underline{E}}^{-1}_{31}\underline{\underline{E}}^{-1}_{32} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 5 & 1 \end{pmatrix}$$

$\underline{\underline{L}}$ ends up being a simpler matrix where the lower triangular part contains the multipliers necessary to eliminate its respective position. The reason why LU decomposition is useful is because triangular systems are easy to solve simply by forward substitution and back-substitution.

$$\underline{\underline{A}}\,\underline{x} = \underline{\underline{L}}\,\underline{\underline{U}}\,\underline{x} = \underline{\underline{L}}\,\underline{y} = \underline{b}$$

First solve $\underline{\underline{L}}\,\underline{y} = \underline{b}$ for $\underline{y}$ and then solve $\underline{\underline{U}}\,\underline{x} = \underline{y}$ for $\underline{x}$. also the determinant is the product of the "pivots", the number you use to eliminate each element below it in

gaussian elimination. These end up on the main diagonal of the upper triangular part $\underline{\underline{U}}$. It can lead to overflows so it's normally calculated as:

$$\ln(\det \underline{\underline{A}}) = \sum_{j=1}^{N} \ln U_{jj}$$

4.2. **Jacobi Method.** Works if diagonally dominant:

$$|A_{ii}| > \sum_{j \neq i} |A_{ij}|$$

$$\begin{pmatrix} \cdot & \cdot & & & & & \\ \cdot & \cdot & \cdot & & & & \\ & \cdot & \cdot & \cdot & & 0 & \\ & & \cdot & \cdot & \cdot & & \\ & 0 & & \cdot & \cdot & \cdot & \\ & & & & \cdot & \cdot & \cdot \\ & & & & & \cdot & \cdot \end{pmatrix}$$

systems with a few variables can often be rearranged into diagonally dominant matrices.

4.2.1. *Method.* :

- $\underline{\underline{A}}\underline{x} = \underline{b}$
- $(\underline{\underline{L}} + \underline{\underline{D}} + \underline{\underline{U}})\underline{x} = \underline{b}$ where L D and U are lower and upper triangular parts that add up to A (as opposed to multiply as in LU decomposition).
- so can find iterative formula $\underline{x}_{n+1} = -\underline{\underline{D}}^{-1}(\underline{\underline{L}} + \underline{\underline{U}})\underline{x}_n + \underline{\underline{D}}^{-1}\underline{b}$

4.2.2. *convergence.* The method always converges if the update matrix $\underline{\underline{T}} = \underline{\underline{D}}^{-1}(\underline{\underline{L}} + \underline{\underline{U}})$:

$$\begin{aligned} [e_i]_{n+1} = [x_i]_{n+1} - x_i &= -T_{ij}[x_j]_n + D_{ij}^{-1}b_j - (-T_{ij}x_j + D_{ij}^{-1}b_j) \\ &= -T_{ij}([x_i]_n - x_i) \\ &= -T_{ij}[\epsilon_i]_n \end{aligned}$$

As before (on page 8):

$$[\zeta_i]_{n+1} = D_{ij}[\zeta_j]_n = \lambda_i[\zeta_i]_n$$

Impose

$$g_i = \left| \frac{[\zeta_i]_{n+1}}{[\zeta_i]_n} \right| = |\lambda_i| \leq 1$$

number of operations is $\mathcal{O}(N^k \times N_{\text{iter}})$ where N_{iter} is the number of iterations required to reach a chosen level of convergence (fractional change is less than a

chosen tolerance), k is 3 for naïve multiplication, but can be less than 3 with magic that isn't taught in this course.

Checking for convergence can be done by finding the fractional change in a norm of the solution.

$$\epsilon = \left| \frac{\|\underline{x}_{n+1}\| - \|\underline{x}_n\|}{\|\underline{x}_n\|} \right|$$

and checking it is below a specified tolerance; something a few orders of magnitude above round off error. Can also sub $\underline{x}_{n+1}$ into $\underline{\underline{A}}\underline{x} = \underline{b}$ and check the fractional difference between LHS and RHS.

4.3. **Iterative Gauss - Seidel Method.** Faster convergence than Jacobi method.

$$\underline{x}_{n+1} = -(\underline{\underline{L}} + \underline{\underline{D}})^{-1}\underline{\underline{U}}\underline{x}_n + (\underline{\underline{L}} + \underline{\underline{D}})^{-1}\underline{b}$$

harder to implement since need to calculate $(\underline{\underline{L}} + \underline{\underline{D}})^{-1}$, (easy to approximate if matrix is diagonally dominant).

4.4. **Eigensystems.**

4.4.1. *Largest Eigenvalue.*

$$\underline{\underline{A}}\underline{x} = \lambda\underline{x}$$

if A is singular then $\exists \underline{x} | \underline{\underline{A}}\underline{x} = 0 \implies$ Nullity$(\underline{\underline{A}}) \neq 0$, so $\lambda = 0$ for those cases in which $\underline{x} \in$ Null$(\underline{\underline{A}})$, so that dim$[E_\lambda(\underline{\underline{A}})]$ = rk$(\underline{\underline{A}})$. If you have a good matrix then $E_\lambda(\underline{\underline{A}})$ completely fills all of $\mathbb{R}^n$, where n is the number of dimensions in which $\underline{x}$ lives.

$$\underline{\underline{A}}^n\underline{v} = \sum_{i=1}^{N} c_i\lambda_i^n\underline{e}_i$$

where $\underline{v}$ is any vector, c_i are the constants which when summed over with the eigenvectors $\underline{e}_i$ will be equal to $\underline{v}$ and λ is the eigenvalues.

$$\lim_{n\to\infty} \underline{\underline{A}}^n\underline{v} \to c_j\lambda_j^n\underline{e}_j$$

so it tends towards the eigenvector with the largest eigenvalue

$$\hat{\underline{e}}_j \approx \frac{\underline{\underline{A}}^n\underline{v}}{\underline{\underline{A}}^n\underline{v}}$$

4.4.2. *Smallest Eigenvalue.* done by finding the largest eigenvalue of $\underline{\underline{A}}^{-1}$, giving λ^{-1}.

4.4.3. *Other Eigenvalues.*

$$\underline{\underline{A}}' := \underline{\underline{A}} - \alpha \underline{\underline{I}}$$

has eigenvalues

$$\underline{\underline{A}}' \underline{e}_i = (\lambda_i - \alpha)\underline{e}_i$$

so finding the largest eigenvalue of $(\underline{\underline{A}}')^{-1}$ gives the closest eigenvalue to α. $(\underline{\underline{A}}')^{-1}$ can be found by solving $\underline{\underline{A}}' \underline{\underline{X}} = \underline{\underline{I}}$ via Jacobi or Gauss-Seidel or LU decomposition. Then for choosing a suitable alpha, consider *Gerschgorin's Theorem*,

$$|\lambda_i - A_{ii}| \leq \sum_{i \neq j} |A_{ij}|$$

which is useful if the matrix is diagonally dominant.

5. Boundary Value Problems

Methods for solving a 2nd order ODE for given boundaries $y(a)$ and $y(b)$, as opposed to $y(a)$ and $y'(a)$.

5.1. **Shooting Method.** start with $y(a) = A$, make a guess of $y'(a) = C_1$ and find a solution $y_1(\eta)$. Which can be obtained via finite difference method (3), or algebraically. The solution won't end at the desired point $y_2(b) = B$ but at $y_1(b) = B_1$ instead.

Make another guess but with $y'(a) = C_2$ instead, and find a new solution $y_2(\eta)$ which ends at $y_2(b) = B_2$. via linearity you can combine the solutions:

$$y_c(\eta) = cy_1(\eta) + (1-c)y_2(\eta)$$

and c is chosen such that $y_c(b) = B$. $y_c(a) = cA + (1-c)A = A$ as required, and $y_c(b) = cB_1 + (1-c)B_2 \overset{!}{=} B$, so $c \overset{!}{=} \frac{B-B_2}{B_1-B_2}$.

5.2. **Finite Differences.** consider

$$\frac{\mathrm{d}^2 y}{\mathrm{d}\eta^2} = k$$

rewrite as CDS approximation:

$$\frac{1}{h^2}(y_{i-1} - 2y_i + y_{i+1}) = k$$

Since $y_{\text{start}} = A$ and $y_m := y_{\text{end}} = B$, the system can be written thusly:

$$\begin{aligned} A - 2y_1 + y_2 &= kh^2 \\ y_1 - 2y_2 + y_3 &= kh^2 \\ &\vdots \\ y_{m-2} - 2y_{m-1} + B &= kh^2 \end{aligned}$$

Using matrices:

$$\begin{pmatrix} -2 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & -2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m-1} \end{pmatrix} = \begin{pmatrix} kh^2 - A \\ kh^2 \\ \vdots \\ kh^2 - B \end{pmatrix}$$

Diagonally dominant system $\implies$ can be solved using Gauss-Seidel or Jacobi method. The result would give a bunch of discrete points that lie within some error to the solution which connects $y(a) = A$ and $y(b) = B$.

5.3. **Derivative Boundary Conditions.** Instead of knowing $y(a) = A$ we know $y'(a) = C$. Take the CDS for the boundary:

$$y_0'(a) \approx \frac{y_1 - y_{-1}}{2h} = C$$

Using the estimate for the 2nd derivative:

$$y_{-1} - 2y_0 + y_1 = kh^2$$

Combine the previous 2 equations:

$$-2y_0 + 2y_1 = kh^2 + 2hC$$

So instead the following system is solved:

$$\begin{pmatrix} -2 & 2 & 0 & 0 & \cdots & 0 & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & -2 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{m-1} \end{pmatrix} = \begin{pmatrix} kh^2 + 2hC \\ kh^2 \\ \vdots \\ kh^2 - B \end{pmatrix}$$

5.4. **Eigensystems.** If the equations are homogeneous and linear, the problem can be viewed as an eigensystem:

$$\frac{\mathrm{d}^2 y}{dx^2} + k^2 y = 0$$

Boundary conditions, $y(0) = 0$ and $y(1) = 0$. (This is a wave equation describing vibrations on a string of length 1 and fixed endpoints). $y(x) = A\sin(kx) + B\cos(kx)$, B.Cs $\implies B = 0,\ k = \pm n\pi$. Using finite differences:

$$\frac{1}{h^2}(y_{i-1} - 2y_i + y_{i+1}) + k^2 y_i = 0$$

$$-y_{i-1} + 2y_i - y_{i+1} = h^2 k^2 y_i$$

Giving

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m-1} \end{pmatrix}$$

Which can also be solved via the aforementioned methods.

6. Minimisation

No fool-proof method for global minima, need to restart algorithms from different regions. Maximisation can be done by applying the same methods to the negative of the f you are interested in.

6.1. **Parabolic Method.** This works when the curvature of f is positive everywhere in the interval. Select 3 points $(x_0, f(x_0))(x_1, f(x_1))(x_2, f(x_2))$ and fit a 2nd-order Lagrange polynomial:

$$P_2(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2$$

The location of the minimum x_3 of the interpolating parabola (from differentiating and setting to zero) is

$$\boxed{x_3 = \frac{1}{2}\frac{(x_2^2-x_1^2)y_0 + (x_0^2-x_2^2)y_1 + (x_1^2-x_0^2)y_2}{(x_2-x_1)y_0 + (x_0-x_2)y_1 + (x_1-x_0)y_2}}$$

The procedure is repeated for the lowest 3 points of $f(x_0), f(x_1), f(x_2), f(x_3)$

6.2. **N-Dimensional Methods.** Different methods shown below.

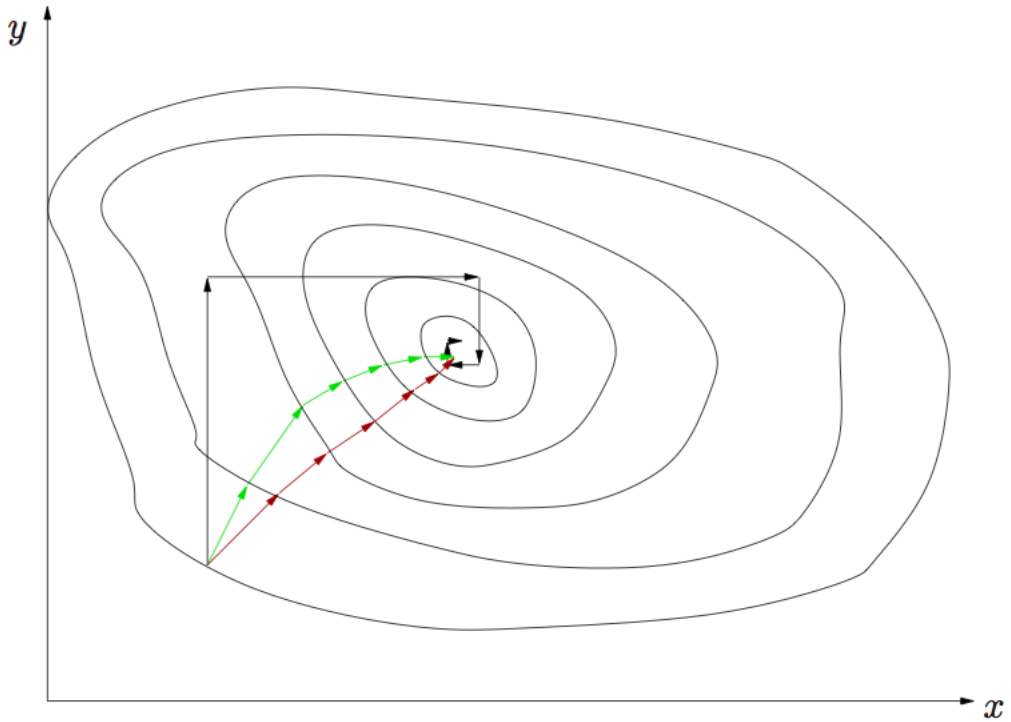


Figure 1. Black arrows show univariate method (slowest), green for Gradient method (faster), red for Newton's method (fastest)

6.2.1. *Univariate Method.* This can be done by searching for the minimum in each direction successively and iterating. This is inefficient.

6.2.2. *Gradient Method.* This is essentially following the steepest descent. This is done by following the gradient of the function in the negative direction:

$$[x_i]_{n+1} = [x_i]_n - \alpha \partial_i(f) \quad \alpha \ll 1$$

where $\partial_i = \frac{\partial}{\partial x_i} \equiv \nabla_i$, for sufficiently small α, $f(x_{n+1}) < f(x_n)$. The gradient needs to be found either by using a finite difference approximation or analytically.

6.2.3. *Newton's Method.* Start at $\underline{x}_0$ that's displaced by $\underline{d}$, the taylor series is then

$$f(\underline{x}_0 + \underline{d}) = f_0 + d_i \partial_i(f_0) + \frac{1}{2} d_i d_j \partial_{ij}(f_0) + \mathcal{O}(|\underline{d}|^3)$$

where $\partial_{ij} := \partial_i \partial_j \equiv H_{ij}$, to be a minimum the Hessian matrix needs to be positive definite, i.e. $d_i d_j \partial_{ij}(f) < 0 \quad \forall \underline{d} \neq 0$. Take the gradient of the 2nd order taylor expansion and setting to 0 gives:

$$\partial_i \left[f_0 + d_j \partial_j(f_0) + \frac{1}{2} d_j d_k \partial_{jk}(f_0) \right] = 0$$

The ∂_i only acts on $\underline{d}$, as $f_0, \partial_j(f_0)$ and $\partial_{ij}(f_0)$ are all constants. Also $\partial_i d_j = \delta_{ij}$.

$$\begin{aligned} \delta_{ij} \partial_j(f_0) + \frac{1}{2} \delta_{ij} d_k \partial_{jk}(f_0) + \frac{1}{2} \delta_{ik} d_j \partial_{jk}(f_0) &= 0 \\ \partial_i(f_0) + \frac{1}{2} d_k \partial_{ik}(f_0) + \frac{1}{2} d_j \partial_{ji}(f_0) &= 0 \end{aligned}$$

Since the Hessian $\partial_{ij} f = \partial_{ji} f$ (i.e. is symmetric), left multiplication is equivalent to right-multiplication so that:

$$d_j \partial_{ij}(f_0) = -\partial_i(f_0)$$

Which can once again be solved for $\underline{d}$ to find the direction and magnitude to iterate towards. If f is exactly parabolic i.e. $f(\underline{x}) = \underline{x}^T \underline{\underline{A}} \underline{x} + \underline{b}^T \underline{x} + c$, then the taylor expansion is exact and $\underline{x}_1 = \underline{x}_0 + \underline{d}$ is exact. If not then we iterate until we get arbitrarily close, $\underline{x}_{n+1} = \underline{x}_n + \underline{d}$, this can be written as

$$\boxed{\underline{x}_{n+1} = \underline{x}_n - [\partial_{ij}(f_n)]^{-1} \partial_i(f_n)}$$

This method is very efficient with quadratic convergence, $|\underline{d}|_{n+1} \sim |\underline{d}|_n^2$, but with large number of dimensions calculating the inverse Hessian can be slow, and using finite differences to approximate the Hessian can lead to a matrix which is not positive definite.

6.2.4. *Quasi-Newton Method.* This introduces a way of approximating $[\partial_{ij}(f_n)]^{-1}$. The formula used is

$$\boxed{\underline{x}_{n+1} = \underline{x}_n - \alpha \underline{\underline{G}}_n \underline{\nabla} f(\underline{x}_n)}$$

Where $\underline{\underline{G}}$ is the approximation and $\alpha \ll 1$. The first approximation $\underline{\underline{G}}_0$ is set to $\underline{\underline{I}}$, which makes the first iteration the same as the gradient search. To update $\underline{\underline{G}}$ use:

$$\underline{\zeta}_n = \underline{x}_{n+1} - \underline{x}_n, \quad \underline{\gamma}_n = \underline{\nabla} f(\underline{x}_{n+1}) - \underline{\nabla} f(\underline{x}_n)$$

update $\underline{\gamma}$ with the gradient of the second order taylor expansion (reusing the calculation from before):

$$\underline{\nabla} f(\underline{x}_{n+1}) \approx \underline{\nabla} f(\underline{x}_n) + \underline{\underline{H}} \underline{\zeta}_n$$

Which can be rearranged to

$$\underline{\underline{H}}_n^{-1} \underline{\gamma}_n = \underline{\zeta}_n$$

We need to update $\underline{\underline{G}}$ to satisfy:

$$\underline{\underline{G}}_n \underline{\gamma}_n = \underline{\zeta}_n$$

So that it mimics the inverse Hessian. The most common method to update this is the Davidon-Fletcher-Power (DFP) algorithm where the update is:

$$\boxed{\underline{\underline{G}}_{n+1} = \underline{\underline{G}}_n + \frac{\underline{\zeta}_n \otimes \underline{\zeta}_n}{\underline{\gamma}_n \underline{\zeta}_n} - \frac{\underline{\underline{G}}_n \underline{\zeta}_n \otimes \underline{\zeta}_n \underline{\underline{G}}_n}{\underline{\gamma}_n \underline{\underline{G}}_n \underline{\gamma}_n}}$$

In Index notation:

$$\boxed{[G_{ij}]_{n+1} = [G_{ij}]_n + \frac{[\zeta_i]_n [\zeta_j]_n}{[\gamma_k]_n [\zeta_k]_n} - \frac{[G_{ip}]_n [G_{qj}]_n [\zeta_p]_n [\zeta_q]_n}{[\gamma_k]_n [\gamma_l]_n [G_{kl}]_n}}$$

The advantage of this scheme is that it only involves multiplications $\mathcal{O}(N^k)$ operations (where $2.37 \leq k < 3$) at each iteration which results in a positive definite Hessian by construction.

7. RNGs

7.1. **Pseudo-Random Numbers.** A *uniform deviate* is a random number lying within a range where any number has the same probability as any other. A simple uniform deviate generator:

$$\boxed{I_{n+1} = (aI_n + c) \bmod m \quad a, c, m \in \mathbb{Z}}$$

where I_n are the uniform deviates, a is a multiplier, c is the increment, and m is the divisor. This generates pseudo-random *integers* in interval $[0, m-1]$. Dividing by m results in a real number:

$$x_n = \frac{I_n}{m} \qquad x \in [0, 1)$$

The sequence of numbers will repeat itself with periodicity m, the periodicity can be much less for poor choices of a, c and m. In general, large m is chosen, with a and c chosen such that the periodicity is equal to m.

Another issue is that there is correlation between successive random numbers. If k random successive numbers are used to plot points in k-dimensional space, they in general will lie in a $k-1$ dimensional plane within k-dimensional space, as opposed to filling every degree of freedom.

7.2. **Transformation Method.** Used for PDFs that are not uniform. Given a normalised uniform distribution:

$$U(x)\mathrm{d}x = \begin{cases} \mathrm{d}x & 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

if $y = f(x)$ then the probabilities transform via:

$$|P(y)\mathrm{d}y| = |P(x)\mathrm{d}x|$$

if $\mathrm{d}y/\mathrm{d}x \geq 0$, the absolute values can be ignored so that

$$\frac{\mathrm{d}x}{\mathrm{d}y} = P(y)$$

Since $P(x) = 1$.
Define:

$$F(y) = \int_{-\infty}^{y} P(y')\mathrm{d}y' = \int_{0}^{x} \mathrm{d}x' = x$$

If F is invertible, x can be mapped into a new variable y which will have the desired PDF.

7.3. **Rejection Method.** First use $f(y)$ called the comparison function, which lies above $P(y)$ for all y. For example, $f(y) = C > P(y) \forall y$.
Procedure:

(1) pick a random number y_i uniformly distributed in $[y_{\min}, y_{\max}]$
(2) pick a random number p_i uniformly distributed in $[0, C]$
(3) If $P(y_i) < p_i$ then reject y_i, and go back to (1), otherwise accept.

Proof. Probability of accepted value in range y and $y + \mathrm{d}y$:

$$\text{Prob} = \frac{\mathrm{d}y}{y_{\max} - y_{\min}} \frac{P(y)}{C} = \text{const} \times P(y)\mathrm{d}y$$

□

Efficiency is obtained by integrating probability of acceptance, which is just the const in the above equation since the PDF is normalised. The inverse of the efficiency is the average number of times (1), (2) and (3) need to be carried out before a value is accepted. The efficiency can be improved by using a comparison function that is closer to the original PDF but is still above it, and also suitable for the transformation method. step (1) becomes, using the transformation method, pick a random deviate y_i in the range between $[y_{\min}, y_{\max}]$, distributed according to the pdf obtained by normalising f.

8. Monte-Carlo Methods

8.1. **Monte-Carlo Integration.**

8.1.1. *Method.* Consider integration in N-dimensions:

$$I = \int_V f(\underline{r})\mathrm{d}^N\underline{r}$$

the integral is related to the mean value of the function within the volume:

$$I = \langle f \rangle V$$

So the integral can be estimated, with random vectors within the region thusly:

$$I \approx \frac{V}{M}\sum_{i=1}^{M} f(\underline{x}_i)$$

8.1.2. *Error.* consider variance in f_i :

$$\sigma_{f_i}^2 = \frac{1}{N-1}\sum_{i=1}^{M}(f(\underline{x}_i) - \langle f \rangle)^2$$

then $\sigma_{\langle f \rangle}^2 = \sigma_{f_i}^2/N$, then the variance in I_{est} is given by:

$$\sigma_{I_{\text{est}}}^2 = V^2\sigma_{\langle f \rangle}^2 = \frac{V^2}{N}\sigma_{f_i}^2$$

then:

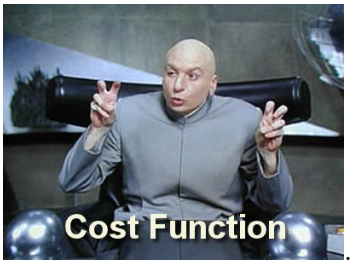
$$I = \frac{V}{M}\sum_{i=1}^{M} f(\underline{x}_i) \pm \frac{V}{\sqrt{N}}\sigma_{f_i}^2$$

8.2. **Monte-Carlo Minimisation - Simulated Annealing.** Useful in cases where:

- Many degrees of freedom.
- many quasi-optimal solutions (local minima) exist in which direct search methods can get stuck instead of finding the global minimum.

We introduce randomness in the search for a minimum. In thermosynamics a system can temporarily move to a less optimal configuration due to fluctuations. The analogy is quantified in the use of the Boltzmann Probability Distribution:

$$P(E)\mathrm{d}E \sim e^{-E/kT}\mathrm{d}E$$



where the energy encodes a . What allows the system to get unstack from local minima is the temperature, allowing for higher energies at a lower probability for even high temperatures. The method uses the *Metropolis Algorithm*, each step the system is changed randomly. The difference in energies before and after the change is calculated. The step is accepted with probability p_{acc} given by:

$$p_{acc} = \begin{cases} 1 & \Delta E \leq 0 \\ e^{-\Delta E/kT} & \Delta E > 0 \end{cases}$$

In functional minimisation, the energy is simply the value of the function and temperature is slowly lowered to 0, hence the analogy with annealing of metals.

9. PDEs

9.1. **classification.**

9.2. **Elliptic.**

9.3. **Hyperbolic.**

9.4. **Parabolic.**

10. Fourier

10.1. **Recap.**

10.2. **Discrete Fourier Transforms.**

10.3. **Sampling Aliasing.**

10.4. **Spectral Methods.**

10.5. **Fast Fourier Transforms.**