Number Representation and Root Finding/Fixed Point Methods

Number Representation
Range: min and max vals that can be represented
precision: min increment b/w adjacent vals
Decimal Representation of 260.25: **1's place is** $10^0/2^0$**!**
$2 \times 10^2 + 6 \times 10^1 + 0 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$
Binary Representation of 260.25: ($2^{10} = 1024$)
$2^8 + 2^2 + 2^{-2} = 100000100.01$
Floating Point representation (scientific notation):
$d.ddddd... \times 10^p$
d is int [0,9]. behind dec. pt is mantissa, p is exponent
BINARY floating pt:
$1.bbbbbbbbbb \times 2^{bbbb}$ (each b is a bit (0 or 1).
REMEMBER exponent is also in binary
IEEE floating pt standard: $f = sMB^{e-E}$
s is sign bit, M is mantissa and leading digit
B is base e is exponent E is bias
Error Sources in Programming
syntax errors
runtime errors (code compiles but fails running)
numerical errors
physics errors
can be systematic or uncontrolled
rounding errors

Basic Root Finding
compute f(x) over mesh, look for sign changes
Bisection root finding:
0)plot and inspect visually
1)Choose interval [a,b] s.t. soln exits to $f(x) = 0$
2)Calculuate first estimate as $(a + b)/2$
3)Determine where soln is (to left or right of midpt.)
4)select new brackets; stop when $|a - b| <$ tolerance.
Secant root finding:
1)Choose two points x_1 and x_2
2)Calculate f_1 and f_2
3)Calculate fire estimate of soln by
extrapolating to x axis with straight line, hitting x_3:
$x_3 = (f_2 x_1 - f_1 x_2)/(f_2 - f_1)$
4) check tolerance
5)update bracket with $x_2 \to x_1$, $x_3 \to x_2$
works even if x_1 and x_2 don't bracket root
converges if $\left|f_{n+1}\right| < \left|f'_n\right| \left|x_{n+1} - x_n\right|$

Newton-Raphson Root Finding
for $f(x) = 0$, write out $f'(x)$
choose some pt x_0 visibly close to root
$x_{n+1} = x_n - f(x_n)/f'(x_n)$
Orders of Convergence of Methods
$\varepsilon_n = x_{n+1} - x_n$ (also can be tolerance).
m is the actual order of conv.
Bisection: $\varepsilon_{n+1} = \varepsilon/2$
Secant: $\varepsilon_{n+1} \simeq \text{const} \times \varepsilon_n^{1.618}$
N-R Method: $\varepsilon_{n+1} \simeq \text{const} \times \varepsilon_n^2$
To determine m, test to see if $\varepsilon_{n+1}/\varepsilon_n^m$ is constant
Will be constant for correct order m
General Fixed Pt Methods
Useful to design methods that converge
Rewrite your needed $f(x)$ as $g(x) = x$
Iteration: $x_{n+1} = g(x_n)$
N-R is a type of F-P iteration
Convergence of FP methods
Converges if in neighborhood of FP, $\left|g'(x)\right| < 1$
(Lipschitz continuous)
So for N-R, converges if $\left|g'(x)\right| = \left|\frac{f(x)f''(x)}{(f'(x))^2}\right| < 1$

Multi-Dimensional Root Finding, Interpolation

for n equations and m unknowns:
$m = n$:
linear in unknowns: use linear alg interp.
nonlinear in unknowns: multi-dim root finding
$m < n$: overdetermined:
linear in unknowns: can use lin alg or least squares
nonlinear in unknowns: nonlinear least sq fitting
Multi Dim Root finding:
Consider system of form:
$F_1(x_1, x_2, ..., x_n) = 0$
$F_2(x_1, x_2, ..., x_n) = 0$
$\vdots$
$F_n(x_1, x_2, ..., x_n) = 0$
Expand fns in Taylor series:
$$F_i(\vec{x} + \delta\vec{x}) = F_i(\vec{x}) + \sum_{j=1}^{n} \frac{\partial F_i}{\partial x_j}\delta x_j + O(\delta x^2)$$
If close to root, we can approximate:
$\vec{F}(\vec{x} + \delta\vec{x}) = \vec{F}(\vec{x}) + \boldsymbol{J} \cdot \delta\vec{x} = 0$
So $\boxed{\vec{x}_{n+1} = \vec{x}_n + \delta\vec{x}}$
$\boldsymbol{J} \cdot \delta\vec{x} = -\vec{F}$
$\boxed{\delta\vec{x} = -\boldsymbol{J}^{-1} \cdot \vec{F}}$
(We're solving for $f_i = 0$)
$\boldsymbol{J} = \left[\frac{\partial \vec{f}}{\partial x_1} \cdots \frac{\partial \vec{f}}{\partial x_n}\right]$
$$\boldsymbol{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$
$\vec{x} = (x_1\ x_2 \cdots x_n)$
Multi-Dim Convergence
Contraction mapping: converges on a FP
after multiple iterations
The surface of bracketing contracts
around FP after each iteration
So, $G(x)$ is a contraction mapping (converges)
to some x_0 if all eigenvals of $\boldsymbol{J}$
are < 1 in abs value
How to:
given some eqn with variables and unknown constants,
plug in known vals for variables to get system of eqns
to solve for consts, (do $f_i = 0$)

Interpolation: regular polynomial
Motivation: functions may be smooth but tough to evaluate
measurements
predictions based upon complicated calculations
Types: Polynomials
Piece-wise polynomials (splines)
Trig functions (Fourier series)
Polynomial Interpolation
Given n points (x,y), they uniquely determine
a polynomial of degree $m = n - 1$
Advantages of using polynomial: smooth,
continuous, infinitely differentiable
Disadvantages: oscillations get crazy if high degree
Process: need to find $f(x) = a_m x^m + a_{m-1}x^{m-1} + \cdots + a_0$
need to find $n = m + 1$ coeffs:
$(x_1, y_1) \quad a_m x_1^m + a_{m-1}x_1^{m-1} + \cdots + a_0 = y_1$
$(x_2, y_2) \quad a_m x_2^m + a_{m-1}x_2^{m-1} + \cdots + a_0 = y_2$
$\vdots$
$(x_n, y_n) \quad a_m x_n^m + a_{m-1}x_n^{m-1} + \cdots + a_0 = y_n$
Lagrange polynomial interpolation:
$$f(x) = \sum_{i=1}^{n} y_i L_i(x) \qquad L_i(x) = \prod_{j=1;j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$
less efficient but more numerically stable
can be applied to regular polynomial fitting
and also splines

Interpolation: Splines
finding interpolating function for each pair of adjacent pts
linear: $f(x) = y_2\frac{(x-x_1)}{(x_2-x_1)} + y_1\frac{(x-x_2)}{(x_1-x_2)}$
continous, nondifferentiable
quadratic: $f_i(x) = a_i x^2 + b_i x + c_i$
conditions: $f_i(x_{i+1}) = y_{i+1}$, $f_i(x_i) = y_i$,
$f'_i(x_{i+1}) = f'_{i+1}(x_{i+1})$
need one more: i.e., use linear interp for 1st interval
continuous and differentiable
cubic: $f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$
$4(n-1)$ coeffs total (n points)
conditions: match functions,
and match first and second derivatives for interior pts
Summary: $f_n(x_n) = f_{n+1}(x_n) = y_x$, $f'_n(x_n) = f'_{n+1}(x_n)$,
$f''_n(x_n) = f''_{n+1}(x_n)$
take second derivative to be zero at endpoints

MISC
Taylor: $\sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$

Sparse Matrices
many more zeroes than not
sparsity: no. zeros/ no. total elements
Inverses usually exist
Advantage: faster to compute
Taylor approx Forward diff:
$f'(x) \simeq (f(x+h) - f(x))/h$ h is mesh
Backward diff:
$f'(x) \simeq (f(x) - f(x-h))/h$
Central diff:
$f'(x) \simeq (f(x+h) - f(x-h))/2h$
h too large: truncation error
h too small? rounding error for 2 very close vals
From Taylor exp,
$f''(x) \simeq (f(x+h) + f(x-h) - 2f(x))/h^2$
Matrix $D^{(2)} = 1/h^2 \times M$
M has diagonals = -2, and elements left
and right of those 1
all others are zero
(but other corners can be 1 if periodic)
Spline Example
Conditions: $f_i(x_i) = y_i$, $f_i(x_{i+1}) = y_{i+1}$
$f_i(x_i) = f_{i+1}(x_i) = y_n$
$f'_i(x_i) = f'_{i-1}(x_i)$, $f''_i(x_i) = f''_{i-1}(x_i)$
You know that $f_i(x) = ax^3 + bx^2 + cx + d$
Can differentiate this as needed

Advantages of different types of root finding
N-R Pros: usually convergent for start points that are good enough
Converges fast, only requires one starting pt
N-R Cons: is derivative hard to do by hand, or may not exist
can be divergent if slope not similar to that near root, such as for oscillatory,
which bisection can be better for
Bisection cons: might not converge if there are poles or an odd number of roots inside bracket, or if not bracketed.

Bilinear Interpolation
make square grid of $\vec{x}$ that brackets chosen point
numbered 1 through 4 CCW from lower left corner
$t = (x - x_1)/(x_2 - x_1)$, $u = (y - y_1)/(y_4 - y_1)$
$z_i \equiv f(\vec{x}_i)$
$f(\vec{x}) = (1-t)(1-u)z_1 + t(1-u)z_2 + tuz_3 + (1-t)uz_4$
Continuous always, but not differentiable at support points (due to sharp discontinuities)

Least-Squares Fitting, Vector Spaces, Matrices, Linear Systems, Eigenproblems

Least-Squares Fitting
More equations (n) than Unknowns (m)
$f(x; a_0, a_1, \cdots, a_m) = y$
Where a_m are the $m+1$ free fit parameters
In general, x, y can be vectors of dimension $n+1$:
this gives $n+1$ equations:
$f(x_i; a_0, a_1, \cdots a_m) = y_i$
Solve for free-fit parameters, minimize residuals:
$S(a_0, a_1, \cdots, a_m) = \sum_{i=0}^{n} [y_i - f(x_i)]^2$
Many physical processes exhibit normal distribution:
Prob density: $\mathcal{P}(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-(y-\mu)^2/2\sigma^2\right\}$
Mean: μ, STDEV: $\sigma^2 \simeq \frac{1}{N-1} \sum_i (y_i - \mu)^2$
Minimizing residuals gives the most probable parameters
Least-sq fitting is a specific example of estimating max of likelihood function
Nonlinear Least-Squares Fitting
$f(x_i, \beta) = y_i$, β are free-fit parameters
Can use Newton-Raphson: $\boldsymbol{J} \cdot \delta\boldsymbol{\beta} = -\boldsymbol{F}$
Levenberg-Marquardt (standard): used in python
$(\boldsymbol{J}^T\boldsymbol{J} + \lambda \text{diag}(\boldsymbol{J}^T\boldsymbol{J}))\delta\boldsymbol{\beta} = -\boldsymbol{J}^T\boldsymbol{F}$
λ is damping factor
need functional form to fit, starting data and parameters guesses

Solving Linear Systems
If det=0, matrix is lin depend
Conditioning: $|\det(A)|/\|A\|$ the larger the better
Condition number $C_A := \left\|A^{-1}\right\| \cdot \|A\|$
$\frac{1}{C_A}\frac{\|\vec{r}\|}{\left\|\vec{b}\right\|} \leq \frac{\|\vec{e}\|}{\|\vec{x}_{true}\|} \leq C_A \frac{\|\vec{r}\|}{\left\|\vec{b}\right\|}$
Useful for estimating magnitude of error
Also, well conditioned if diagonal dominant (magnitude of each diagonal greater than sum of magnitudes of other row elements)
residual $\vec{r} = A\vec{x} - \vec{b}$
error $\vec{e} = \vec{x} - \vec{x}_{true}$
$A\vec{x} = \vec{b} \qquad \vec{x} = A^{-1}\vec{b}$
Lower triangular: forward substi:

$$x_m = \left(b_m - \sum_n a_{mn} x_n\right) / a_{mm}$$

m goes from first row to last row, the sum
sums up all of the previous iterations
Upper triangular: backward substi:
m begins at lowest row. n iterates from $m+1$ to 1 (not 0).
Gaussian Elimination
Steps to get to upper triangular form:
1) Use 1st row (pivot) to eliminate the first element
of each of the lower rows in sequence
2) Use 2nd row (pivot) to elim 2nd element of each of the lower rows
etc

No. operations: $\sum_{i=1}^{n-1} (2n - (2i-1))(n-i)$

DETAILED STEPS:
1) Take R_1, divide by a_{00}, multiply by a_{10}
2) Subtract that from R_2
Do same to lower row:
3) Use R_1 and divided by a_{00}, multiply by new a_{20}
etc...
Algorithm:

```
for i in range(0,n-1):
    for j in range(i+1,n):
        c = A[j,i]/A[i,i]
        A[j,i] = 0.0
        A[j,i+1:n] -= c*A[i,i+1:n]
        b[j] = b[j]-c*b[i]
```

LU Decomposition
$A = LU$ L,U triangular matrices
Find with GE: U is matrix from GE
L is matrix of multipliers (`c`) in code used to
eliminate each element, put in place where that element was.
Advantage over GE: less computationally intensive.
Furthermore with multiple b's and same A, you'll need
to do full GE for every b, but with LU, only needed once.
One GE is O(N^3), one trigular solving is O(N^2).
LU only requires one of each of these for many b's

Vector Spaces, Vectors, Matrices
A set that is closed over finite vector addition and scalar multiplication
Examples of vector spaces and their basis sets:
Cartesian $(\hat{i}, \hat{j}, \hat{k})$
Polynomials $(1, x, x^2 \cdots)$
Set of variables to root find over
Scalars form a vector field
Vector spaces defined over field
Inner Products/Dot Products
$\vec{v} \cdot \vec{w} = (\vec{w} \cdot \vec{v})^*$
$a\vec{v} \cdot \vec{w} = a(\vec{v} \cdot \vec{w})$
$(\vec{v} + \vec{u}) \cdot \vec{w} = \vec{v} \cdot \vec{w} + \vec{u} \cdot \vec{w}$
Two vectors orthogonal if inner product is zero
polynomials like Legendre polys result from orthogonalization
Linear operator: $A(\vec{x} + \vec{y}) = A\vec{x} + A\vec{y}$
and scalar property
and 2 real non-colinear vectors form a basis of $\mathbb{R}^2$
Dimension: of space V, number of vectors in a basis
More on linear operators
always map $\vec{0}$ onto $\vec{0}$
kernel: set of vectors $\vec{x}$ such that:
$\ker(f) = \{\vec{x} \in V : f(\vec{x}) = 0\}$
image of f: set of output vectors within output space W.
$\dim(\ker(f)) + \dim(\text{im}(f)) = \dim(V)$
Inverse: $M^{-1}Mv = v$
Inverse exists if $\ker(M) = 0$, $\det(M) \neq 0$,
rows of M form set of linearly indep vecs
$\det(AB) = \det(A)\det(B)$, $\det(kA) = k^n \det(A)$
Matrix Norms

Euclidean Norm: $\|A\| \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2}$

$\|aM\| = |a|\|M\|$, $\|A\vec{v}\| \leq \|A\|\|\vec{v}\|$
Triangle Inequality: $\|A + B\| \leq \|A\| + \|B\|$

Eigenvalue Problems
$A\vec{w} = \lambda\vec{w}$
$A\vec{x} = \vec{b}$ take form $A\vec{w} = \lambda\vec{w}$
$(A - \lambda I)\vec{w} = \vec{0} \qquad \det((A - \lambda I)) = 0$
Certain ODEs/systems can be solved by eigendecomposition:
Assume soln has form $u(x) = ce^{iwt}$
$\sum F_i = m_i\ddot{x}_i = -k_i(x_i - x_{x-1}) - k_{i+1}(x_i - x_{i+1})$
Can use this to write out matrix as
$A\vec{x} = -\omega^2\vec{x} = \ddot{\vec{x}}$
$\vec{x}(t) = \sum_j C_{\pm j}\vec{w}_j e^{\pm i\omega_j t}$
For this, two eigenvectors then give four solns, accounting for $\pm$ in constants and in exponent
for example, eigenvalues are $-\omega^2$
Eigendecomposition
$A\vec{w} = \lambda\vec{w}$
let $\vec{w} = P\vec{w}^*$
so $AP\vec{w}^* = \lambda P\vec{w}^*$,
$A^* = \lambda\vec{w}^*$, $A^* = P^{-1}AP$
Need to find P st A^* diagonal.
Steps: Find eigendecomp P (like Jacobi method)
Find A^*, eigenvals are diagonals of A^*
eigenvecs are columns of P

Iterative Solvers
(all of my written x's are vectors)
given A and b
Direct solvers: exact solution if no liminations for precision
(finite processes)
Iterative solvers:
not exact answer, but we have idea for how good
Iterative improvement
x_0 is true soln: $Ax_0 = b$, x is guess
$Ax = b + r$ r is residual, $x_0 = x - \delta x$
$Ax = A(x_0 + \delta x) = b + r$, $r = A\delta x$, $\delta x = A^{-1}r$
$x_{i+1} = x_i - \delta x$
Jacobi Method
$Ax = b \qquad (L + D + U)x = b$
L is elements below diag NOT TRIANGULAR; U are above, D is diag.
$x_{n+1} = -D^{-1}(L + U)x_n + D^{-1}b$
Convergence: for multiple dimensions, FP method needs to
be a contraction mapping to converge $(x_{n+1} = G(x_n))$
Converges to x_0 if all eigvals of J of G are ¡1 in abs val
similar to 1D: $\left|g'(x)\right| < 1$
Easier (but not always necessary): converges if A is row diagonal dominant
Gauss-Seidel: $x_{n+1} = -(L + D)^{-1}Ux_n + (L + D)^{-1}b$
still comes from $A = L + D + U$
Increasing Basin of Convergence via Relaxation
Rewrite Jacobi as weighted sum:
$x_{n+1} = \omega(-D^{-1}(L + U)x_n + D^{-1}b) + (1 - \omega)x_n$
only convergent where $0 < \omega < 2$
Under-relaxed if $0 < \omega 1$: interpolated b/w previos soln to new soln
increases basin, calms down oscillatory iterations
Over-relaxed if $1 < \omega 2$: extrapolates from previous soln to beyond new soln
Can therefore give faster conv, but negatively affect basin
to derive these, get two diff x's and then make one you xn, one xn+1

Integration

Numerical Integration
Why?: most integrals can't be solved analytically
Often just want numbers, not analytic solns
especially true for high dimensions
automation and speed often more important than analyticity
Rectangle Method:
mesh points range from 0 to n, h is mesh spacing

$$I = \sum_{i=0}^{n-1} f(x_i)(x_{i+1} - x_i) = h \sum_{i=0}^{n-1} f(x_i)$$

Midpoint Method:

$$I = h \sum_{i=0}^{n-1} f\left([x_i + x_{i+1}]/2\right)$$

Trapezoidal Method:

$$I = h \sum_{i=0}^{n-1} f\left([f(x_i) + f(x_{i+1})]/2\right)$$

SIMPLIFIED Trapezoidal Method:

$$I = \frac{h}{2}(f_0 + f_n) + h \sum_{i=1}^{n-1} f_i$$

Simpson's Methods:
Use polynomials to approximate the function in small interval
$f(x) \simeq p(x) = \alpha + \beta(x - x_i) + \gamma(x - x_i)(x - x_{i+1})$
$\alpha = f_i, \quad \beta = (f_{i+1} - f_i)/h, \quad \gamma = (f_i - 2f_{i+1} + f_{i+2})/2h^2$

$$I_i = \int_{x_i}^{x_{i+2}} p(x)\,\mathrm{d}x$$

$I_i = \frac{h}{3}(f_i + 4f_{i+1} + f_{i+2})$
So, **Simpson's 1/3 Method:**

$$I = \frac{h}{3}\left[f_0 + 4 \sum_{i=1,3,5\ldots}^{n-1} f_i + 2 \sum_{i=2,4,6\ldots}^{n-2} f_i + f_n\right]$$

Simpson's 3/8 (cubic) Method:
$f(x) \simeq p(x) = \alpha + \beta(x - x_i) + \gamma(x - x_i)(x - x_{i+1}) + \delta(x - x_i)(x - x_{i+1})(x - x_{i+2})$
gives $I_i = \frac{3h}{8}(f_i + 3f_{i+1} + 3f_{i+2} + f_{i+3})$

$$I = \frac{3h}{8}\left[f_0 + 3 \sum_{i\%3=1}^{n-1} f_i + 3 \sum_{i\%3=2}^{n-2} f_i + 2 \sum_{i\%3=0}^{n-3} f_i + f_n\right]$$

Summary: in order of rectangular/midpoint, trap., 1/3, and 3/8:
interpolation is nearest neighbor, linear, quad, cubic;
no. of points needed to define each I_i: 1, 2, 3, 4.

Improper Integrals
Have upper or lower limit of ∞
Has singularity at or b/w upper and lower lims
Cannot be evaluated at or b/w upper and lower ims
Is otherwise integrable (convergent)
Limit at singularity: ex:

$$\int_0^2 \frac{1}{x^p}\,\mathrm{d}x$$

converges if $p < 1$
Limit at infinity, not over singularity: ex:

$$\int_2^\infty \frac{1}{x^p}\,\mathrm{d}x$$

converges if $p > 1$
How to evaluate:
Strategy 1: (semi-infinite interval (a or b inf)) substitute $y = 1/x$
Strategy 2:(two inf limits) break into two integrals, then use strat 1
Strategy 3: (power law singularity at end of interval)
substitute $y = (x - a)^{1-\gamma}, 0 \leq \gamma < 1$

Gaussian Quadrature
Limitations of previous numerical integration methods:
Formula: $\int_a^b f(x)\,\mathrm{d}x \simeq \sum_{i=0}^{n} A_i f(x_i)$
x_i are abscissas, A_i are weights, $w(x)$ is weighting function
Underlying assumption that function can be expanded in a basis of polynomials $p_j(x)$

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{j=0}^{2n+1} c_j \int_a^b p_j(x)\,\mathrm{d}x$$

EXACTLY reproduced by $\sum_{i=0}^{n} A_i f(x_i)$
The size of missing higher-order terms governed by quality of polynomial expansion
More generally: $\int_a^b F(x)\,\mathrm{d}x = \int_a^b w(x)f(x)\,\mathrm{d}x$
Weighting function can be singular; $f(x)$ is polynomial-like
$f(x)$ can be decomposed into a set of orthogonal polynomials
Gaussian quadrature is exact if $f(x)$ is a lin comb of
polynomials $p_j(x)$, $j_{max} \leq 2n + 1$, that are **orthogonal across the weighting function**

$$\int_a^b w(x)p_m(x)p_n(x)\,\mathrm{d}x = 0, \quad m \neq n$$

Absicissas are the $n + 1$ roots of $p_{n+1}(x) = 0$

STEPS
1) Choose $w(x)$ and $f(x)$ based off of original integrand $F(x)$:

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

2) Choose an orthogonal basis set $p_j(x)$ across $w(x)$
As needed, modify oringla integral to put into form assumed by basis set
3) Choose the order of quadrature n then determine weights A_i and abscissas x_i:
A) look up
B) Use a generating function?
C) Use principles that abscissas are zeros of $p_{n+1}(x)$
4) Compute integral:

$$\int_a^b w(x)f(x)\,\mathrm{d}x \simeq \sum_{i=0}^{n} A_i f(x_i)$$

Gauss Chebyshev: $A_i = \pi/(n + 1)$, $x_i = \cos\left(\frac{(2i+1)\pi}{2n+2}\right)$

Fourier Stuff and ODEs

Dirac Delta

$$\delta(\omega) = \int_{-\infty}^{\infty} \exp(i\omega t)dt = \begin{matrix} 0, & \omega \neq 0 \\ \infty, & \omega = 0 \end{matrix}$$

Continuous FT

Decomposiion of some $f(t)$ into $e^{i\omega t}$ basis set

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{i\omega t}\,\mathrm{d}t$$

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{-i\omega t}\,\mathrm{d}t$$

Discrete Fourier Transform (DFT)

Define finite interval [-L,L]

Define basis set as cos and sine functions that have int no. of oscillations over the interval

Discrete Fourier decomp:

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k\cos\left(\frac{k\pi x}{L}\right) + b_k\sin\left(\frac{k\pi x}{L}\right))$$

$$a_k = \frac{1}{L}\int_{-L}^{L} f(x)\cos\left(\frac{k\pi x}{L}\right)\mathrm{d}x \qquad b_k = \frac{1}{L}\int_{-L}^{L} f(x)\sin\left(\frac{k\pi x}{L}\right)\mathrm{d}x \qquad a_0 = \frac{1}{2L}\int_{-L}^{L} f(x)\,\mathrm{d}x$$

From continuous to discrete

$$H_n = \Delta\sum_{k=0}^{N-1} h_k e^{2i\pi kn/N}$$

Aliasing and Nyquist Frequency

How fine should our mesh be to capture a function properly?

In the discretized form, each freq amplitude is contaminated with other frequencies

The result cannot be undone from the discrete values alone.

ODEs

Rewriting nth order ODE as coupled 1st order

Method 1: facorization ex:

$y'' - 2y' - 3y = f(x) \Rightarrow [D-3][D+1]y = f(x)$

Let $y_1 = y$, $y_2 = [D+1]y$

So $y_2 = [D+1]y_1 = y_1' + y_1$, $[D-3]y_2 = f(x) = y_2' - 3y_2$

Method 2: Brute Force

Let $y_1 = y$, $y_2 = y'$

IVP fixed step integrators

Given $y'(x) = f(x, y)$ and $y(x_0) = y_0$

Compute y on a mesh $y_i = y(x_i)$ with step size h

Euler's Method

Use truncated Taylor expansion $y(x+h) = y(x) + y'(x)h + O(h^2)$

$y(x+h) = y(x) + f(x, y(x))h + O(h^2)$

Error accumulation over n steps: $O(h)$.

Continued →

Using Fourier Transforms to solve ODEs
Ex: DDO $m\ddot{x} + \gamma\dot{x} + kx = f(t)$
If $f(t) = Ae^{i\omega_0 t}$, $x(t) = c_1 e^{i\omega_0 t}$, $c_1 = \frac{A}{-m\omega_0^2 + i\gamma\omega_0 + k}$
General soln: $X(\omega) = \frac{F(\omega)}{-m\omega^2 + i\gamma\omega + k}$
Next, consider the orthogonal basis set of the Fourier decomp: $v_j = e^{i\omega_j t}$
$\frac{\mathrm{d}}{\mathrm{d}t} v_j = i\omega_j v_j \quad \rightarrow \quad Av_j = \lambda_j v_j$
Fourier basis are eigenfunctions of the differential operator
So, any system $A\vec{x} = \vec{b}$ can be written as
$x = \sum_j x_j v_j \quad b = \sum_j b_j v_j \quad A \sum_j x_j v_j = \sum_j b\lambda_j x_j v_j = \sum_j b_j v_j$
Now you have a set of independent equations: $\sum_j x_j = \sum_j b\frac{b_j}{\lambda_j}$
Can be generalized to any linear system with known eigenfuncs of operator
Stiff Differential Equations and Implicit Methods
ODE Eigenval Method Ex: Consider system
$u' = 998u + 1998v \quad u(0) = 1$
$v' = -999u - 1999v \quad v(0) = 0$
$\begin{bmatrix} u' \\ v' \end{bmatrix} = D \begin{bmatrix} u \\ v \end{bmatrix} = \lambda \begin{bmatrix} u \\ v \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 998 & 1998 \\ -999 & -1999 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \lambda \begin{bmatrix} u \\ v \end{bmatrix}$
$\det\{A - I\lambda\}$ gives eigenvals λ_1, λ_2, eigenvecs $\vec{w}_1, \vec{w}_2$
$\begin{bmatrix} u(x) \\ v(x) \end{bmatrix} = c_1 \vec{w}_1 e^{\lambda_1 x} + c_2 \vec{w}_2 e^{\lambda_2 x}$
PROBLEMS: Matrix ill-conditioned so determinant could result in error.
Need step size to be $h \leq 2/\lambda$
from $y' = -\lambda y \quad y_{n+1} = y_n + hy'_n = (1 - \lambda h)y_n$
Having two totally diff eigenvals mean you can't compute both simultaneously
Eigenvals stiff if very different
REMEDY: use Implicit methods
$y_{n+1} = y_n + hy'_{n+1} = y_n + \lambda h y_{n+1} = \frac{y_n}{1+\lambda h}$
STABLE FOR $\forall h$ especially large h, useful for solutions over large time
Can be generalized to coupled ODEs
$\vec{y}' = -C \cdot \vec{y} \quad \vec{y}_{n+1} = (I - Ch) \cdot \vec{y}_n$
$(I - Ch)^n \rightarrow 0$ for $\lambda_1 max < 1$
$\vec{y}_{n+1} = (I + Ch)^{-1} \vec{y}_n$
THIS would be useful for solving that original example.
Can also write above (to avoid matrix inversion) with 1st order Taylor exp as:
$\vec{y}_{n+1} = \vec{y}_n + h \left[I - h\frac{\partial \vec{f}(\vec{y}_{n+1})}{\partial \vec{y}} \right]^{-1} \vec{f}(\vec{y}_n)$
This can generalize to higher orders :)