Fast Fourier Transform (FFT)

If f(x) is a periodic function with period T [f(x+T)=f(x)], then it can be represented as a Fourier series

$$f(x)=\frac{a_0}{2}+\sum_{n=1}^{\infty}\left[a_n\cos\left(\frac{2\pi n}{T}x\right)+b_n\sin\left(\frac{2\pi n}{T}x\right)\right],$$

or in complex form,

$$f(x)=\sum_{n=-\infty}^{\infty}c_n\exp\left(-\frac{2\pi i\, n}{T}x\right),$$

where $a_n=\frac{1}{2}\Re\, c_n;\;\; b_n=\frac{1}{2}\Im\, c_n;\;\; c_{-n}=c_n^*.$

Multiplying by exp(2π*imx*/*T*) and integrating,

$$c_m=\frac{1}{T}\int_0^T f(x)\exp\left(\frac{2\pi i\, m}{T}x\right)dx$$

$$f(x)=\sum_{n=-\infty}^{\infty} c_n \exp\left(-\frac{2\pi i\, n}{T}x\right)=\sum_{n=-\infty}^{\infty} c_n \exp\left(-i\,\omega_n x\right),\quad \omega_n=\frac{2\pi n}{T}.$$

Fourier series have many nice properties.

$$f'(x)=\sum_{n=-\infty}^{\infty} -i\,\omega_n c_n \exp\left(-i\,\omega_n x\right).$$

Fourier series coefficients of the derivative are obtained by multiplying those of the function by -iω_n. Similarly, for the integral, divide by -iω_n.

Convolution $(f*g)=\frac{1}{T}\int_0^T f(x)\,g(y-x)\,dx.$ $\qquad c_n^{(f*g)}=c_n^{(f)}\,c_n^{(g)}$

Besides obvious applications to signal processing, Fourier analysis has many other applications.

E.g., equation $f''(x)+f(x)=q(x),$ where q(x) is given, turns into

algebraic equations $\left(-\omega_n^2+1\right)c_n^{(f)}=c_n^{(q)}.$ Spectral methods for solving PDEs.

Computing Fourier coefficients efficiently is important.

$$c_n = \frac{1}{T}\int_0^T f(x)\exp\left(\frac{2\pi i\, n}{T}x\right)dx$$

Note if f(x) is periodic with period T, the same is true for the whole integrand.

We know that the trapezoidal rule is very accurate for periodic integrands.

If we divide the interval [0,T] into N subintervals, then $x_j = jT/N$, $h = T/N$.

$$c_n = \frac{1}{T}\cdot\frac{T}{N}\left[\frac{1}{2}f_0 + f_1\exp\left(\frac{2\pi i\, n\times 1}{N}\right) + f_2\exp\left(\frac{2\pi i\, n\times 2}{N}\right) + \ldots + f_{N-1}\exp\left(\frac{2\pi i\, n\times[N-1]}{N}\right) + \frac{1}{2}f_N\right].$$

But $f_N = f_0$, so

$$c_n = \frac{1}{N}\sum_{j=0}^{N-1} f_j\exp\left(\frac{2\pi i\, n\, j}{N}\right). \qquad C_n = Nc_n = \sum_{j=0}^{N-1} f_j\exp\left(\frac{2\pi i\, n\, j}{N}\right).$$

Note that $$C_{n+N} = \frac{1}{N}\sum_{j=0}^{N-1} f_j\exp\left(\frac{2\pi i\,(n+N)\, j}{N}\right) = \frac{1}{N}\sum_{j=0}^{N-1} f_j\exp\left(\frac{2\pi i\, n\, j}{N} + 2\pi i\, j\right) = c_n$$

The coefficients in this approximation are periodic with period N. Only N independent coefficients, e.g., $C_{-N/2+1}$, ..., $C_{N/2}$ for even *N* or $C_{-[N/2]}$, ..., $C_{[N/2]}$ for odd *N*. In fact, for real data only [N/2]+1 are independent (C_0, ..., $C_{[N/2]}$). Makes sense: index N/2 corresponds to the Nyquist frequency, which is the highest frequency that can be represented given the sampling rate.

4 $$C_n = \frac{1}{N}\sum_{j=0}^{N-1} f_j \exp\left(\frac{2\pi i n j}{N}\right), \quad n = -N/2+1,\ldots,N/2 \quad \text{or} \quad -(N-1)/2,\ldots,(N-1)/2.$$

This solves the problem in principle, but calculating these coefficients straightforwardly is inefficient. *N* terms in each sum, *N* sums $\Rightarrow N^2$ operations.

Can we do better?

Before that, a couple of things to mention. The expression above is called the discrete Fourier transform. Transforms N numbers f_j into N numbers, each of which is a linear combination of f_j.

More than just an approximation of the integral. Suppose we have the set of values of function f(x) at equidistant points xj: fj=f(xj). Interpolate by a trigonometric polynomial of degree N-1:

$$f_j = \frac{1}{N}\sum_{n=-(N-1)/2}^{(N-1)/2} C_n \exp\left(-\frac{2\pi i j}{N} n\right), \quad N \quad \text{odd},$$

$$f_j = \frac{1}{N}\left[c_{N/2}\cos(\pi j) + \sum_{n=-N/2-1}^{N/2-1} c_n \exp\left(-\frac{2\pi i j}{N} n\right)\right], \quad N \quad \text{even}$$

(Just as for regular polynomial interpolation, for f(x) the series is infinite, for a discrete sample it has to be cut off to be unambiguous).

Then C_n are given **exactly** by the equation above.

5 $C_n = \sum_{j=0}^{N-1} f_j \exp\left(\frac{2\pi i n j}{N}\right), \quad n = -N/2+1, \ldots, N/2 \quad \text{or} \quad -(N-1)/2, \ldots, (N-1)/2.$

Discrete convolution $(x * y) = \sum_{j=0}^{N-1} x_j y_{k-j}$

$$C_n^{(x*y)} = C_n^{(x)} C_n^{(y)}$$

This has applications, e.g., for calculating the coefficients of the product of two polynomials.

$$P^{(1)}(x) = \sum_{k=0}^{L} a_k x^k \qquad P^{(2)}(x) = \sum_{k=0}^{M} b_k x^k$$

$$P^{(1)}(x) \times P^{(2)}(x) = \sum_{k=0}^{L+M} d_k x^k \qquad d_k = \sum_{j=0}^{k} a_j b_{k-j}$$

A slightly different problem, but related.

Already mentioned the nodes of the Clenshaw-Curtis rule.

Lots of applications.

Fast Fourier Transform

$$C_n = \frac{1}{N}\sum_{j=0}^{N-1} f_j \exp\left(\frac{2\pi i n j}{N}\right), \quad n = -N/2+1,\ldots,N/2 \quad \text{or} \quad -(N-1)/2,\ldots,(N-1)/2.$$

Will be convenient to switch to n=0,...,N.

It is possible to calculate all C_N in O(NlogN) operations instead of $O(N^2)$.

There are now algorithms that can do this for any N (worst case: prime numbers), but the simplest and fastest algorithm is for $N=2^m$.

The algorithm was (re)discovered by Cooley and Tukey in 1965 and became popular after that publication, but was first found by Gauss in ~1805 (!) and a few people afterwards.

One of the "top 10 algorithms of the 20th century" (CiSE, Jan.-Feb. 2000)

Roughly $2N^2$ vs. $1.5N\log_2 N$ operations. $(2N^2)/(1.5N\log_2 N)=(4/3)N/\log_2 N$.

A factor of ~100 already for $N = 2^{10} = 1024$.

Very sophisticated free library: FFTW (Fastest Fourier Transform in the West)

Works for any N, automatically chooses between many different algorithms taking processor architecture into account. http://www.fftw.org/

$$C_n = \sum_{j=0}^{N-1} f_j \exp\left(\frac{2\pi i n j}{N}\right), \quad n = 0,\ldots N.$$

Consider N = 2^3 = 8

$$\begin{aligned}
C_0 &= f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 \\
C_1 &= f_0 + f_1 e^{i\pi/4} + i f_2 + f_3 e^{3i\pi/4} - f_4 - f_5 e^{i\pi/4} - i f_6 - f_7 e^{3i\pi/4} \\
C_2 &= f_0 + i f_1 - f_2 - i f_3 + f_4 + i f_5 - f_6 - i f_7 \\
C_3 &= f_0 + f_1 e^{3i\pi/4} - i f_2 + f_3 e^{i\pi/4} - f_4 - f_5 e^{3i\pi/4} + i f_6 - f_7 e^{i\pi/4} \\
C_4 &= f_0 - f_1 + f_2 - f_3 + f_4 - f_5 + f_6 - f_7 \\
C_5 &= f_0 - f_1 e^{i\pi/4} + i f_2 - f_3 e^{3i\pi/4} - f_4 + f_5 e^{i\pi/4} - i f_6 + f_7 e^{3i\pi/4} \\
C_6 &= f_0 - i f_1 - f_2 + i f_3 + f_4 - i f_5 - f_6 + f_7 \\
C_7 &= f_0 - f_1 e^{3i\pi/4} - i f_2 - f_3 e^{i\pi/4} - f_4 + f_5 e^{3i\pi/4} + i f_6 + f_7 e^{i\pi/4}
\end{aligned}$$

Note that in all equations, terms of the same colour either have the same coefficient or differ by sign only. So, e.g., instead of 8 combinations of f_1 and f_5 only need to calculate two, f_1+f_5 and f_1-f_5, and use each of them 4 times (multiplying by different factors).

$$f_0 \pm f_4, \quad f_1 \pm f_5, \quad f_2 \pm f_6, \quad f_3 \pm f_7$$

$$C_n = \sum_{j=0}^{N-1} f_j \exp\left(\frac{2\pi i n j}{N}\right), \quad n = 0,\ldots N.$$

$$f_0 \pm f_4, \quad f_1 \pm f_5, \quad f_2 \pm f_6, \quad f_3 \pm f_7$$

These are actually DFTs for N=2: $C_0 = f_0 + f_1$; $C_1 = f_0 - f_1$

$$\begin{aligned}
C_0 &= f_0 + f_1 & +f_2 + f_3 & & +f_4 + f_5 & & +f_6 + f_7 \\
C_1 &= f_0 + f_1 e^{i\pi/4} & +if_2 + f_3 e^{3i\pi/4} & & -f_4 - f_5 e^{i\pi/4} & & -if_6 - f_7 e^{3i\pi/4} \\
C_2 &= f_0 + if_1 & -f_2 - if_3 & & +f_4 + if_5 & & -f_6 - if_7 \\
C_3 &= f_0 + f_1 e^{3i\pi/4} & -if_2 + f_3 e^{i\pi/4} & & -f_4 - f_5 e^{3i\pi/4} & & +if_6 - f_7 e^{i\pi/4} \\
C_4 &= f_0 - f_1 & +f_2 - f_3 & & +f_4 - f_5 & & +f_6 - f_7 \\
C_5 &= f_0 - f_1 e^{i\pi/4} & +if_2 - f_3 e^{3i\pi/4} & & -f_4 + f_5 e^{i\pi/4} & & -if_6 + f_7 e^{3i\pi/4} \\
C_6 &= f_0 - if_1 & -f_2 + if_3 & & +f_4 - if_5 & & -f_6 + f_7 \\
C_7 &= f_0 - f_1 e^{3i\pi/4} & -if_2 - f_3 e^{i\pi/4} & & -f_4 + f_5 e^{3i\pi/4} & & +if_6 + f_7 e^{i\pi/4}
\end{aligned}$$

Likewise, only 4 (not 8) different combinations of 4 terms. These are DFTs for N=4: $C_0 = f_0 + f_1 + f_2 + f_3$; $C_1 = f_0 + if_1 - f_2 - if_3$; $C_2 = f_0 - f_1 + f_2 - f_3$; $C_3 = f_0 - if_1 - f_2 + if_3$. Each is a combination of the N=2 DFTs above.

Calculate 4 N=2 DFTs (4×2 = 8 numbers): $f_0 \pm f_4,\ f_1 \pm f_5,\ f_2 \pm f_6,\ f_3 \pm f_7$

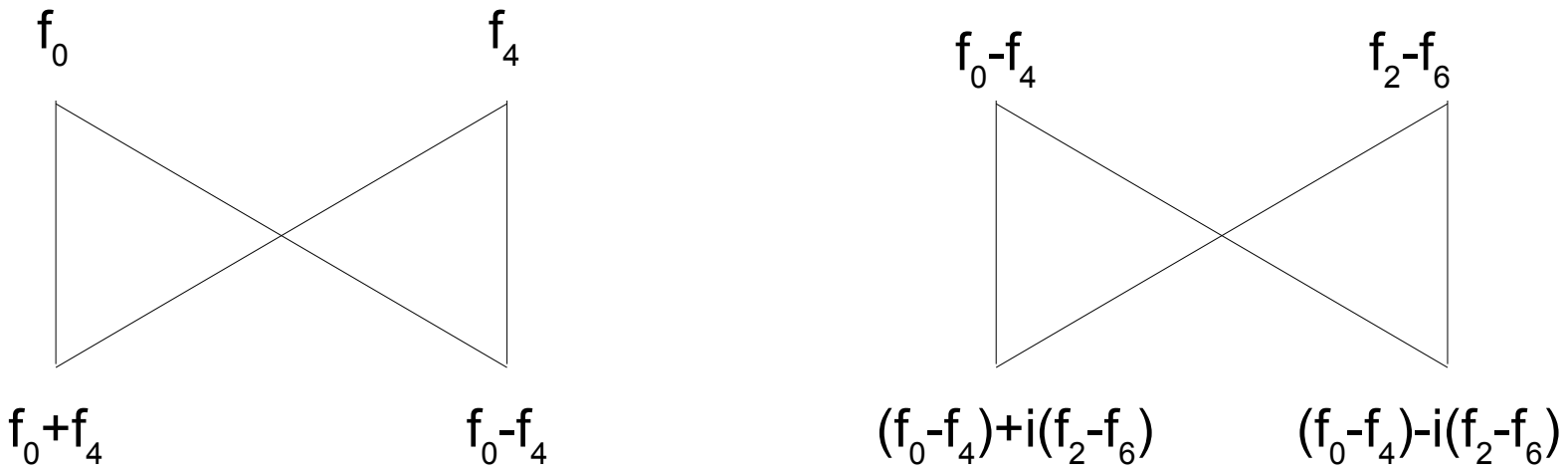
Use them to calculate 2 N=4 DFTs (2×4 = 8 numbers). Note that each of the N=2 DFT numbers is combined with only one other N=2 DFT number, but twice.

For instance, f_0+f_4 is combined with f_2+f_6 twice:

$$(f_0+f_4)+(f_2+f_6) \text{ and } (f_0+f_4)-(f_2+f_6);$$

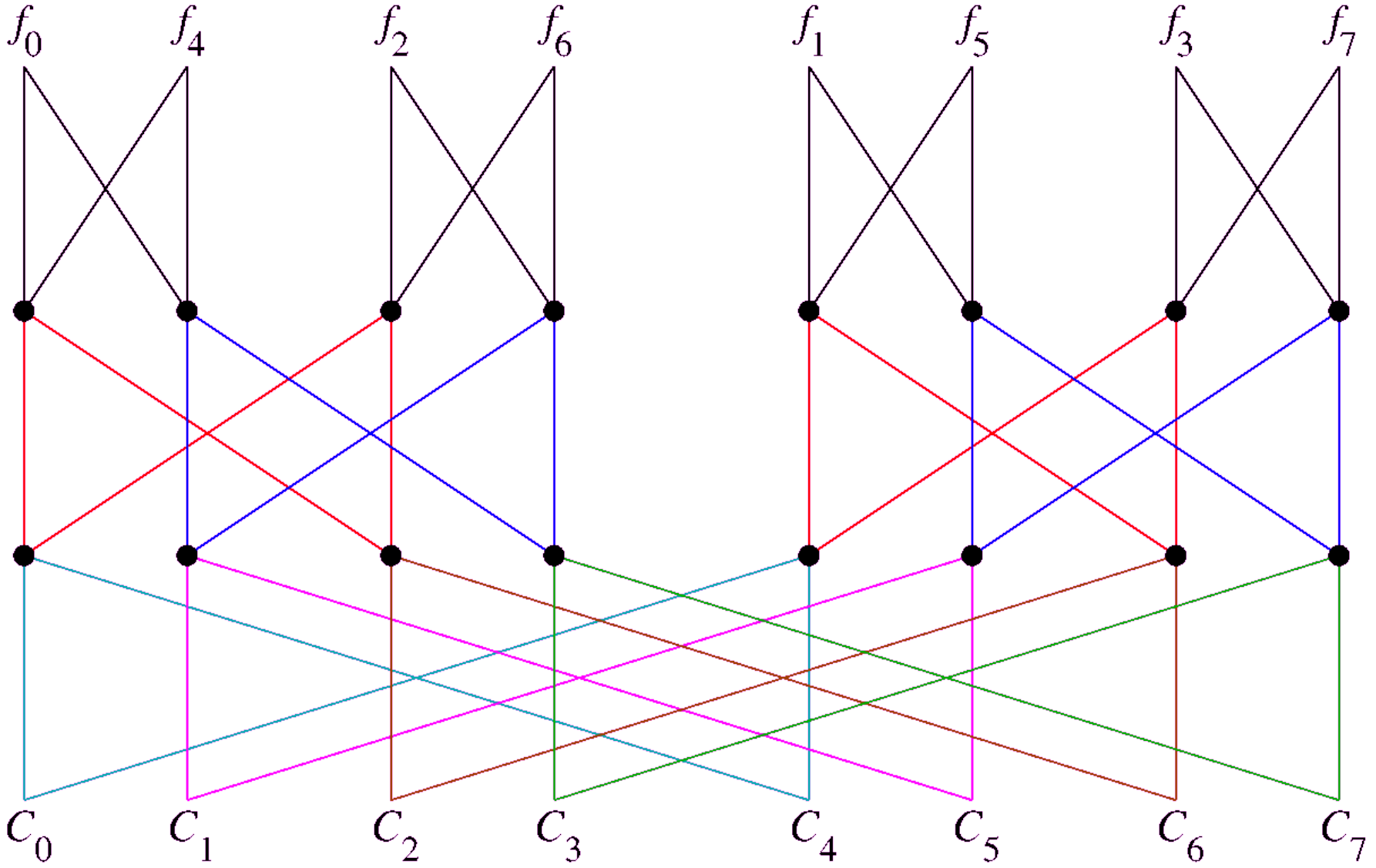
$$(f_0-f_4)+i(f_2-f_6) \text{ and } (f_0-f_4)-i(f_2-f_6);$$

Such combinations can be represented as "butterflies":



Each "butterfly" corresponds to (at most) 3 operations: 1 multiplication, 1 addition, 1 subtraction. Finally, 8 N=4 DFT #s are combined to calculate 8 N=8 DFT #s.

“Butterfly diagram”



4 = N/2 “butterflies” in each level. 3 operations in each “butterfly”. 3 = $\log_2$ N levels. Total operation count is (3N/2)$\log_2$ N. Neglects some O(N) ops.

$$C_n = \sum_{j=0}^{N-1} f_j \exp\left(\frac{2\pi i n j}{N}\right) =$$

$$\sum_{j=0}^{N/2-1} f_{2j} \exp\left(\frac{2\pi i n(2j)}{N}\right) + \sum_{j=0}^{N/2-1} f_{2j+1} \exp\left(\frac{2\pi i n(2j+1)}{N}\right) =$$

$$\sum_{j=0}^{N/2-1} f_{2j} \exp\left(\frac{2\pi i n j}{N/2}\right) + \sum_{j=0}^{N/2-1} f_{2j+1} \exp\left(\frac{2\pi i n j}{N/2}\right) \exp\left(\frac{2\pi i n}{N}\right) =$$

$= C_n^e + W^n C_n^o$ $\quad$ C_n^e is the DFT of order N/2 using f_j with even j.

$W = \exp(2\pi i / N)$ $\quad$ C_n^o is the DFT of order N/2 using f_j with odd j.

Note n runs from 0 to N-1. It is understood that $C^e_{n+N/2} = C^e_n$; $C^o_{n+N/2} = C^o_n$

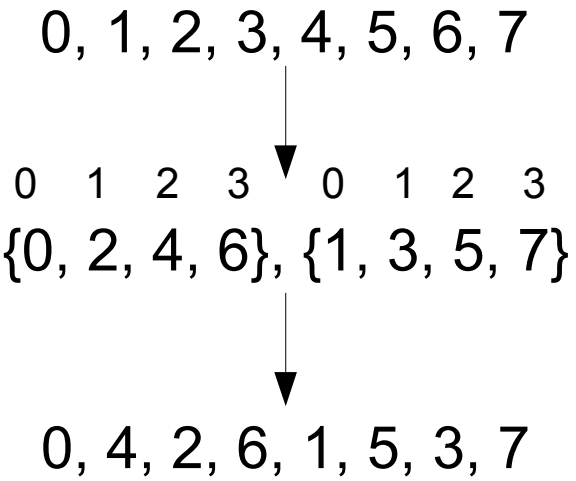
$$C_{n+N/2} = C_n^e + W^{n+N/2} C_n^o = C_n^e - W^n C_n^o$$

Indeed, each C_n^e is combined with only one C_n^o , but twice, once with "+" sign and once with "-" sign. This is the "butterfly". 3 operations; assumes that W^n are pre-computed [but this is O(n)].

Of course, DFTs with N/2 can then be expressed in terms of those with N/4, etc.

Initial ordering of f_j

Even and odd should be in different halves, and then in each half, when numbered in increasing order, even and odd should again be in different halves, etc.



Binary:

0 → 000, 1 → 001, 2 → 010, 3 → 011, 4 → 100, 5 → 101, 6 → 110, 7 → 111.

Parity is determined by the least significant bit (LSB) – should be far apart. Within each group, parity is determined by the next LSB, etc. Those that differ by the MSB only should be next to each other. Do bit reversal and then order.

0 → 000 → 000 → 0; 1 → 001 → 100 → 4; 2 → 010 → 010 → 2; 3 → 011 → 110 → 6; 4 → 100 → 001 → 1; 5 → 101 → 101 → 5; 6 → 110 → 011 → 3; 7 → 111 → 111 → 7

13 Can be modified straightforwardly to calculate the inverse DFT, i.e., f_j from C_n.

$$f_j = \sum_{n=0}^{N-1} C_n \exp\left(-\frac{2\pi i n j}{N}\right)$$

In practice, often have to deal with functions that are not periodic or whose period is unknown. We can do a Fourier transform

$$f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} C(k)\exp(-ikx)\,dk \qquad C(k) = \int_{-\infty}^{\infty} f(x)\exp(ikx)\,dx$$

Numerically, have to consider a finite interval: $C(k) \sim \int_0^T f(x)\exp(ikx)\,dx$

But the integrand is not periodic. Effectively, a discontinuity at the boundary, which produces $C(k) \sim k^{-1}$. Will drown out lower-intensity high-frequency components.

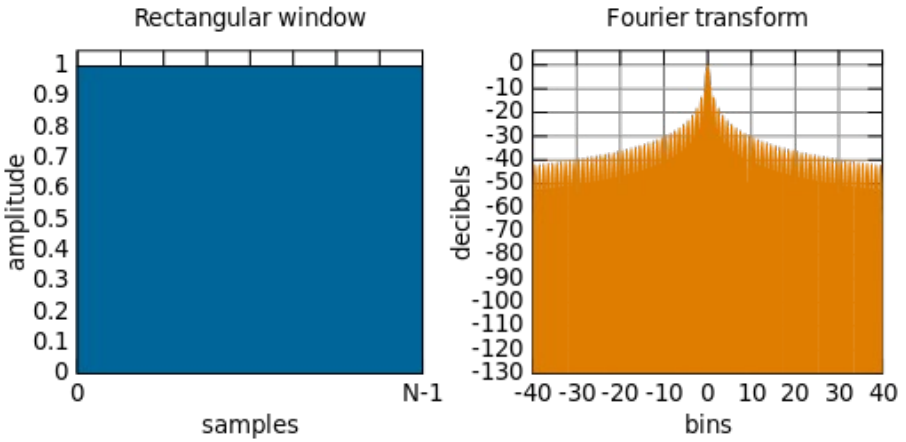
A solution: apply a window function

$$C(k) \approx \int_0^T W(x) f(x)\exp(ikx)\,dx$$

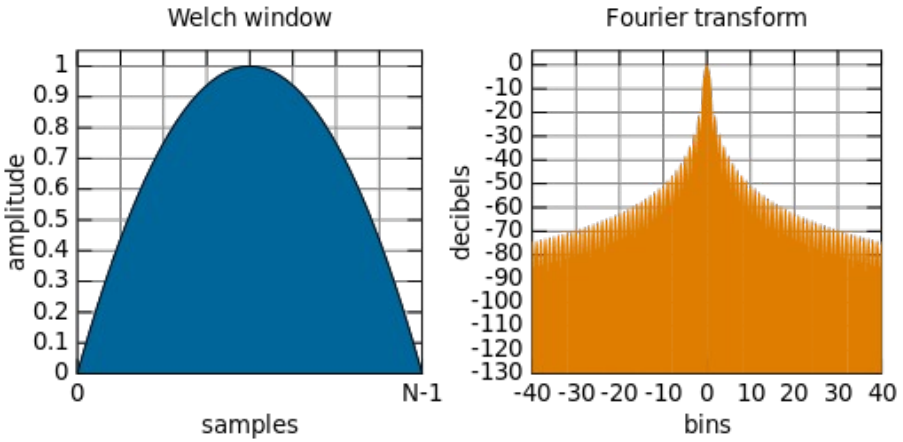
$$C(k) \approx \int_{-\infty}^{\infty} W(x) f(x) \exp(ikx) dx$$

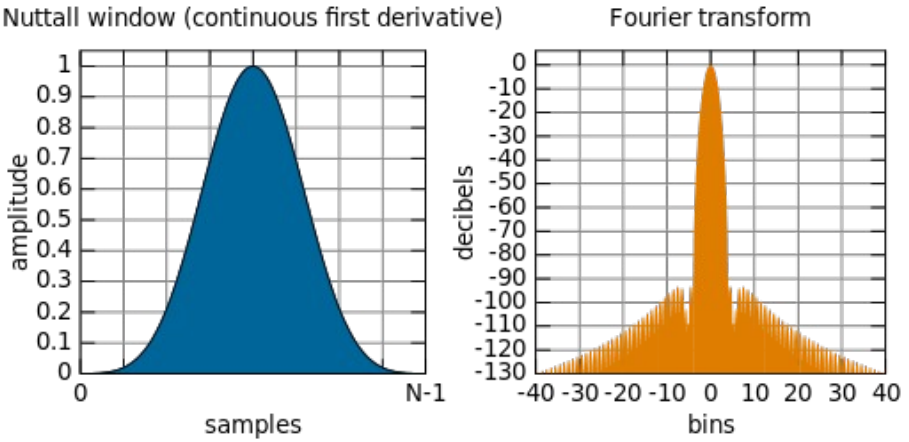
$$\int_0^T f(x) \exp(ikx) dx$$

corresponds to the rectangular W(x)



Other functions decrease more gently towards the edges of the interval.





http://en.wikipedia.org/wiki/Window_function has about 20 window functions

Generally, the tradeoff is between the width of the central peak and the intensity of the sidelobes.

Solving nonlinear equations

For a given f(x), find x such that f(x) = 0

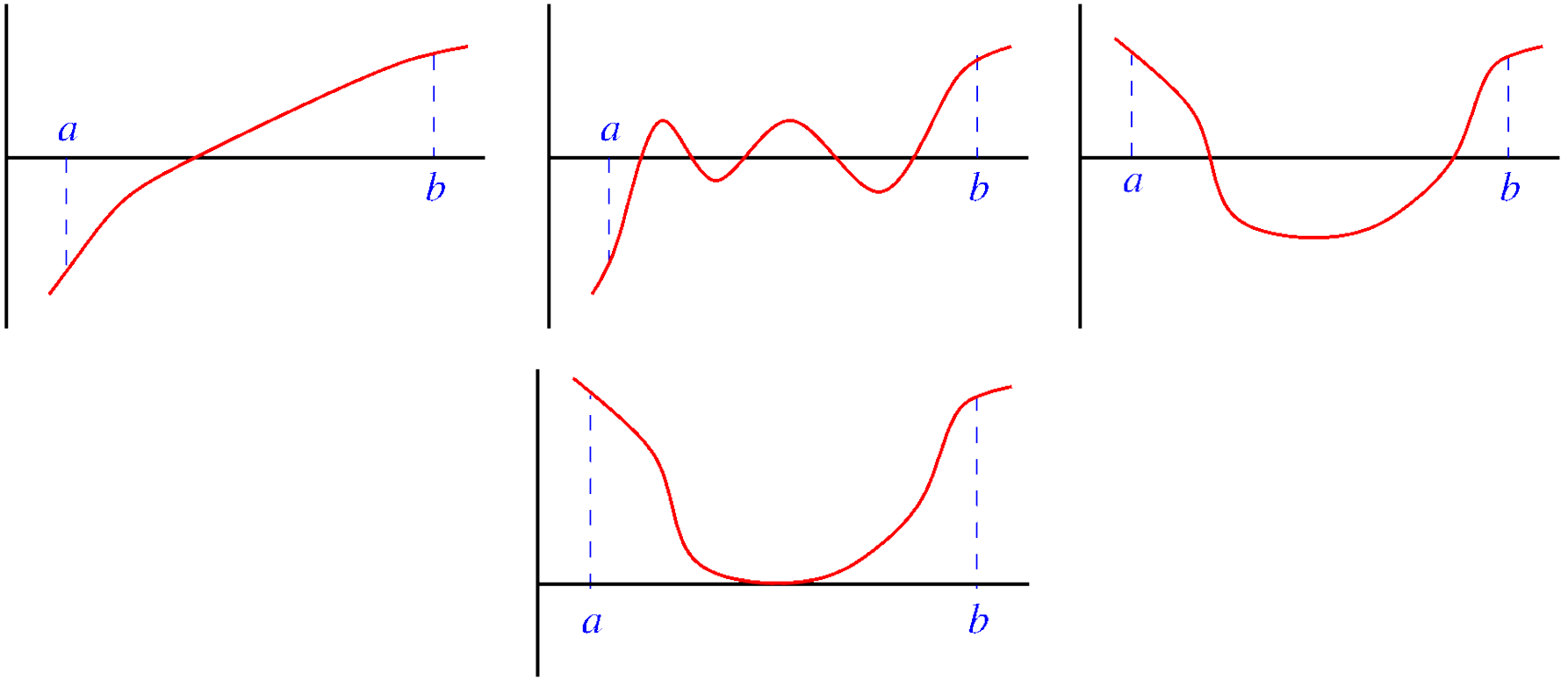
Also known as root finding

Ideally, would like a method that would find all roots for any f(x). No such method exists. All methods require either an interval in which the solution is to be found or an initial guess and most methods will find at most one root.

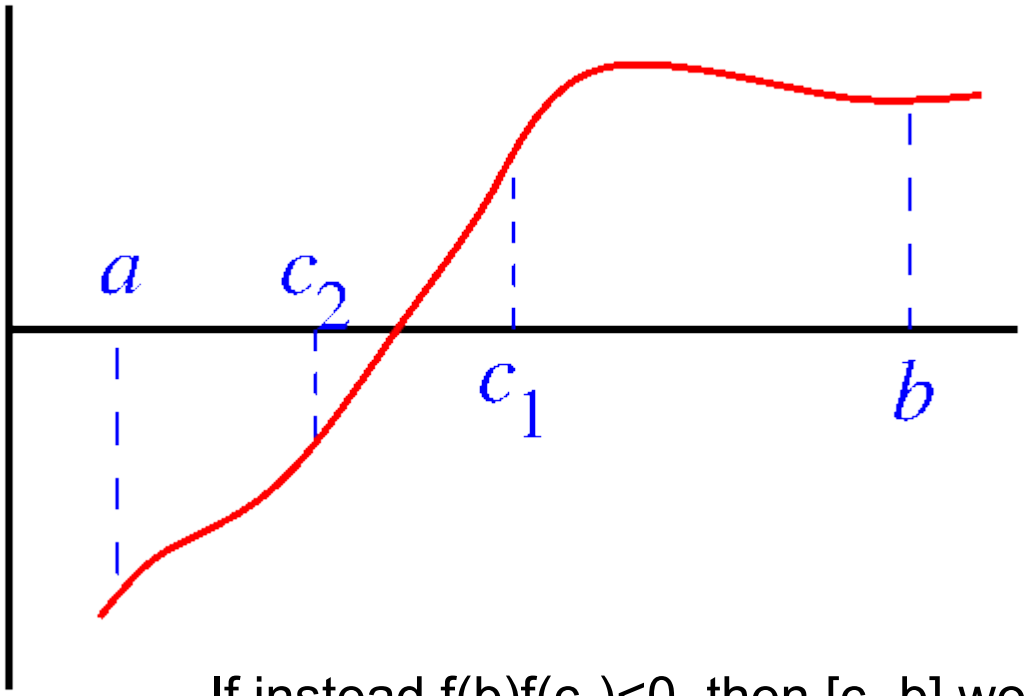
Possible sources:

1. Plotting the function.
2. Solving an approximate equation analytically.
3. Using the solution of a slightly different equation (e.g., if f(x) depends on some parameter *y* and the goal is to study how the solution depends on *y*, then, after solving for a particular value of *y*, one can use that solution as an initial guess for a slightly different value of *y*.

If for a continuous function f, f(a) < 0 and f(b) > 0, then there must be at least one root between a and b. Divide a large interval into small subintervals and pick those where the signs are opposite at the ends. Obviously, there can still be more than one root in such a subinterval. Worse, there can be situations where f(a)f(b) > 0, but there are still roots inside. In the case of, e.g., a double root, no interval [a,b] containing the root with f(a)f(b)<0 exists.



Nevertheless, if an interval with f(a)f(b) < 0 is found, bisection method is guaranteed to find a solution. (If there are many solutions, it will find one of them.)



Define $c_1 = (a+b)/2$.

If $f(a)f(c_1)<0$ (as in the figure), then b is replaced with c_1 and the new interval $[a,c_1]$ is considered. It is known with certainty that there is a root in this interval, so it remains bracketed.

If instead $f(b)f(c_1)<0$, then $[c_1,b]$ would be considered.

The new interval $[a,c_1]$ is bisected again: $c_2 = (a+c_1)/2$. Since $f(c_1)f(c_2) < 0$, the new interval is $[c_1,c_2]$. The process continues until the width of the interval is smaller than the required tolerance.

The uncertainty of the root is the width of the interval.

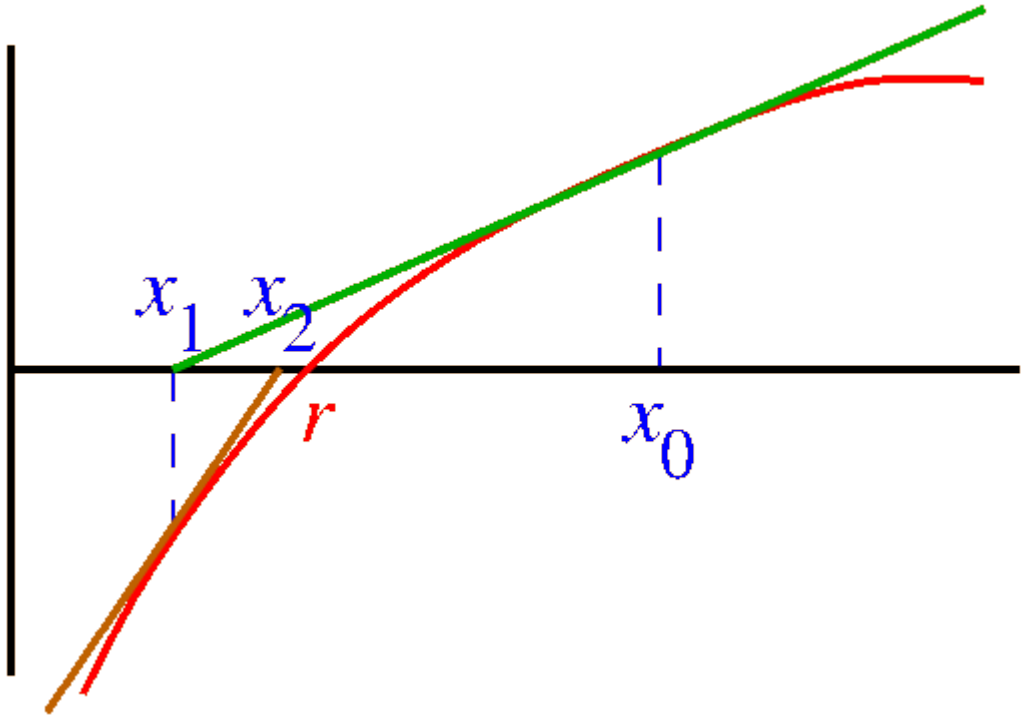
Since the interval width decreases by a factor of 2 at each step, then the uncertainty

$$\epsilon_{n+1} = \frac{1}{2}\epsilon_n$$

When $\varepsilon_{n+1} = \alpha\varepsilon_n$, the method is said to converge linearly. (This is a bit confusing, because $\epsilon_n \sim e^{-n\ln\alpha}$.) In many areas that would be considered fast, but not in root finding. There are faster methods, but the advantage of bisection is reliability (and simplicity).

Newton (or Newton-Raphson) method

Not a bracketing method. Starts with a point, not an interval.



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

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

$$\epsilon_i = x_i - r$$

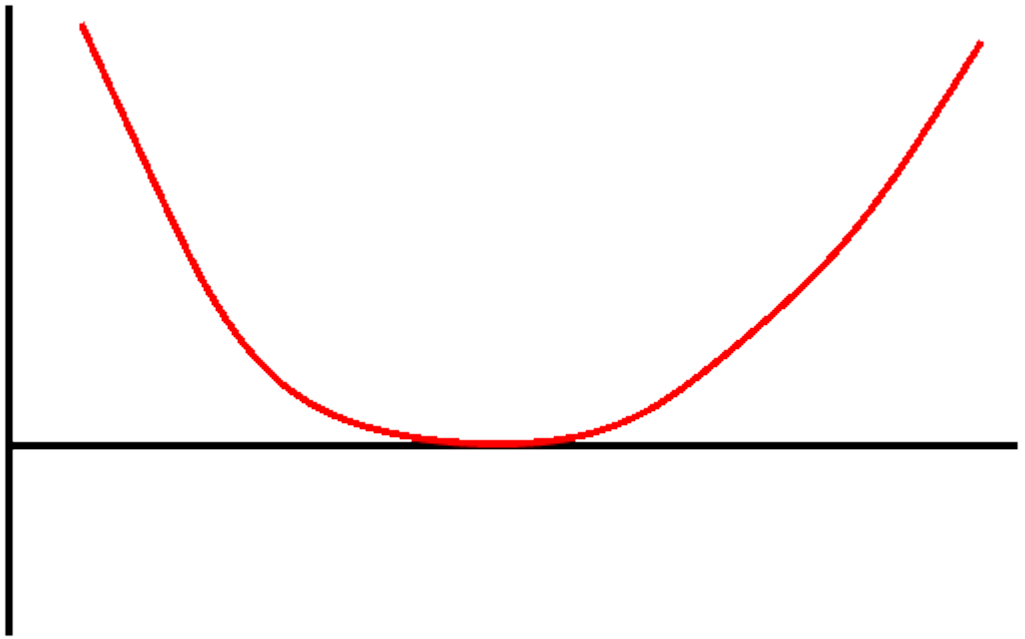
$$\epsilon_{i+1} = \epsilon_i - f(x_i)/f'(x_i)$$

$$0 = f(r) = f(x_i) - f'(x_i)\epsilon_i + (1/2)f''(x_i)\epsilon_i^2 + \ldots \Rightarrow \frac{f(x_i)}{f'(x_i)} \approx \epsilon_i - \frac{f''(x_i)}{2f'(x_i)}\epsilon_i^2$$

$$\epsilon_{i+1} = \frac{f''(x_i)}{2f'(x_i)}\epsilon_i^2$$

Quadratic convergence. Number of significant digits approximately doubles every step. When the function is linear, converges in 1 step. But if f ' is 0, diverges immediately.

Unlike bisection, the Newton-Raphson method is not guaranteed to converge. But it can find roots that bisection cannot find, e.g., double roots.



$$\epsilon_{i+1} = \frac{f''(x_i)}{2f'(x_i)}\epsilon_i^2$$

$$f''(x_i)/f'(x_i) \propto \epsilon_i^{-1}$$

$$\epsilon_{i+1} \propto \epsilon_i$$

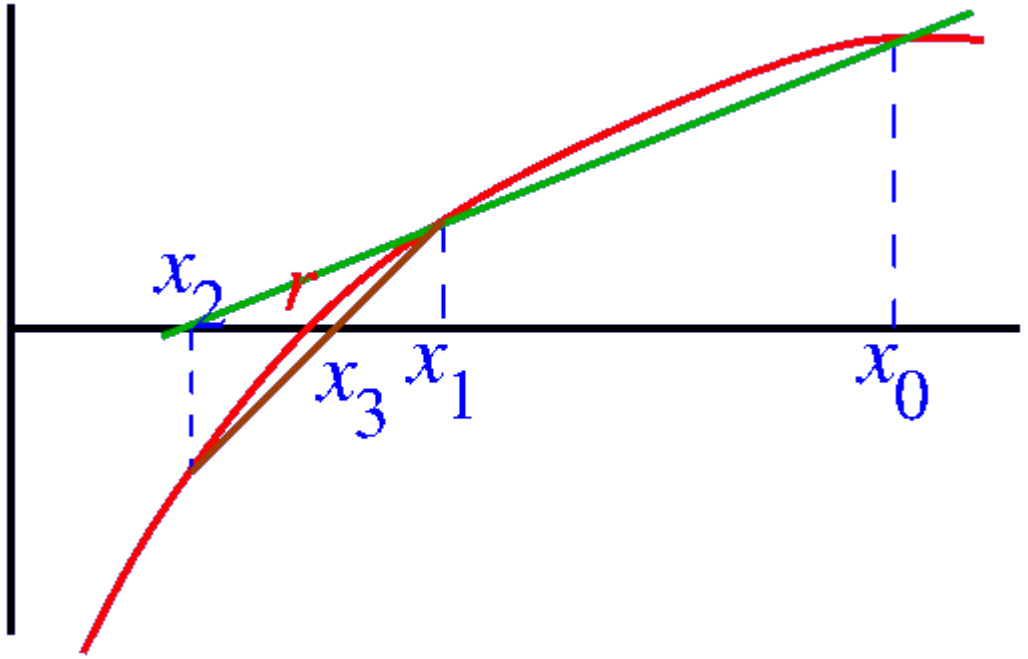
Linear convergence.

If it is known that the root is multiple, can speed up convergence. If $f(x) \approx C(x-r)^m$, then $f'(x) \approx Cm(x-r)^{m-1}$. $f(x)/f'(x) \approx (x-r)/m$.

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

Another possibility is considering roots of function f(x)/f'(x).

Newton-Raphson method requires the derivative. If it cannot be calculated analytically, use the secant method.



$$x_{n+1} = x_n - f(x_n)\left[\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}\right]$$

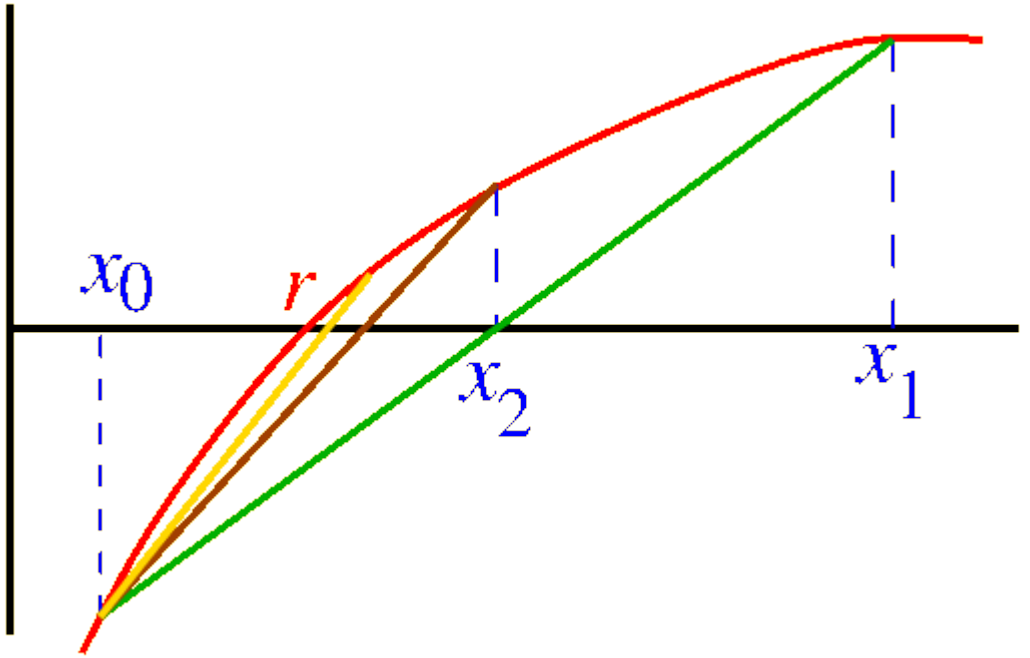
[...] looks like the numerical derivative

$$\epsilon_{n+1} \approx \frac{f''(r)}{2f'(r)}\epsilon_n \epsilon_{n-1}$$

If we assume that $\epsilon_{n+1} \propto \epsilon_n^{\alpha}$, then $\epsilon_n^{\alpha} \propto \epsilon_n \epsilon_n^{1/\alpha} \Rightarrow \alpha = 1 + 1/\alpha \Rightarrow \alpha = \frac{1+\sqrt{5}}{2} \approx 1.62$.

Faster than linear, but slower than quadratic.

There is a variant that ensures bracketing, called false position (or regula falsi) method



Point x_{n+1}, instead of always replacing x_{n-1}, replaces the point where the function has the same sign.

Very often, a point is preserved forever; the interval width does not shrink to zero; the slope does not approach the derivative at r, so the method has linear convergence (can be faster than bisection).

But there are also more sophisticated methods that both preserve bracketing and ensure quadratic convergence whenever possible, switching to bisection otherwise. Brent method.

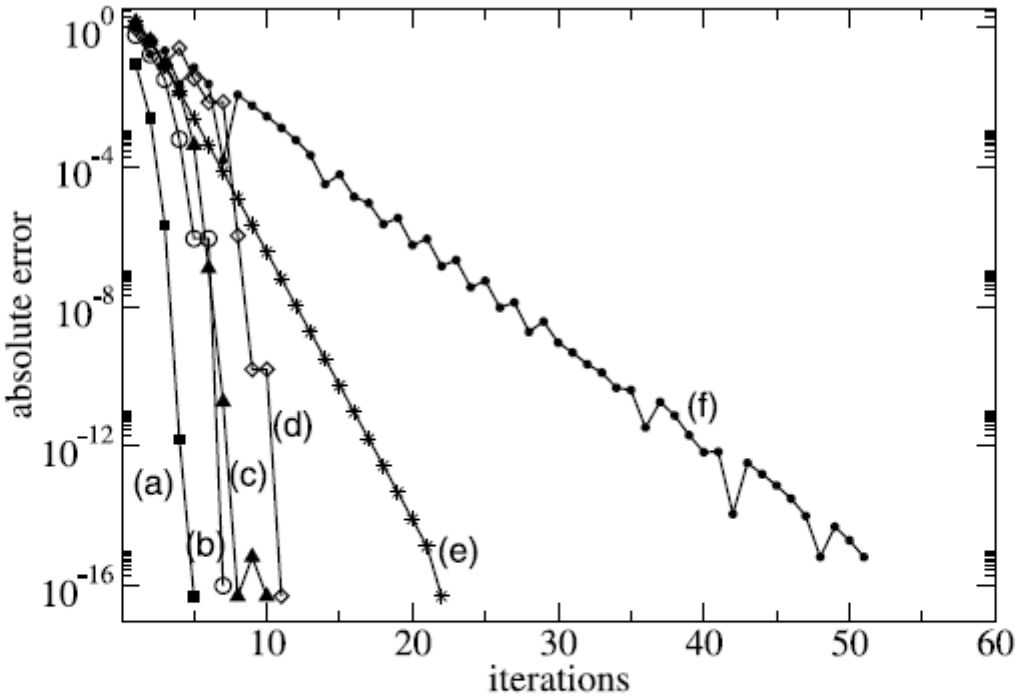
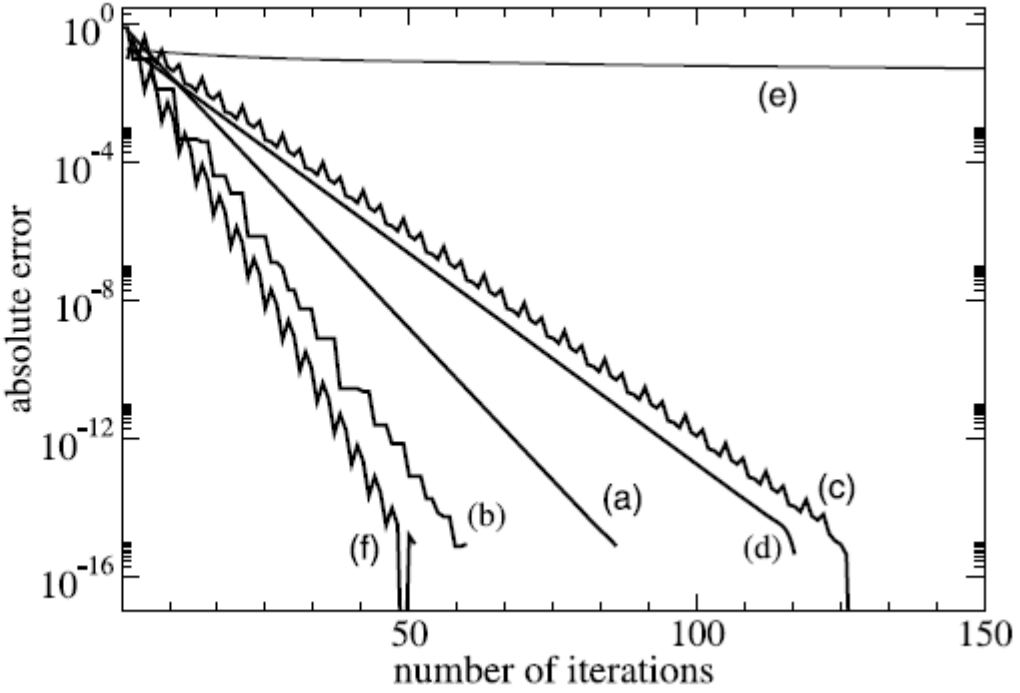


Fig. 6.10 (Comparison of different solvers) The root of the equation $f(x) = x^2 - 2$ is determined with different methods: Newton-Raphson (*a*, *squares*), Chandrupatla (*b*, *circles*), Brent (*c*, *triangle up*), Dekker (*d*, *diamonds*), regula falsi (*e*, *stars*), pure bisection (*f*, *dots*). Starting values are $x_1 = -1, x_2 = 2$. The absolute error is shown as function of the number of iterations. For $x_1 = -1$, the Newton-Raphson method converges against $-\sqrt{2}$

Simple root

Fig. 6.11 (Comparison of different solvers for a third order root) The root of the equation $f(x) = (x-1)^3$ is determined with different methods: Newton-Raphson (*a*), Chandrupatla (*b*), Brent (*c*), Dekker (*d*), regula falsi (*e*), pure bisection (*f*). Starting values are $x_1 = 0$, $x_2 = 1.8$. The absolute error is shown as function of the number of iterations



Triple root.

Roots of polynomials

Same methods can be applied, although there are also special methods. Easier, because we know how many roots there are. One useful trick is deflation of polynomials – divide by (x-r) after root r is found. This way, cannot converge to the same root again.

Solving sets of nonlinear equations

$$\vec{f}(\vec{x}) = \vec{0}$$

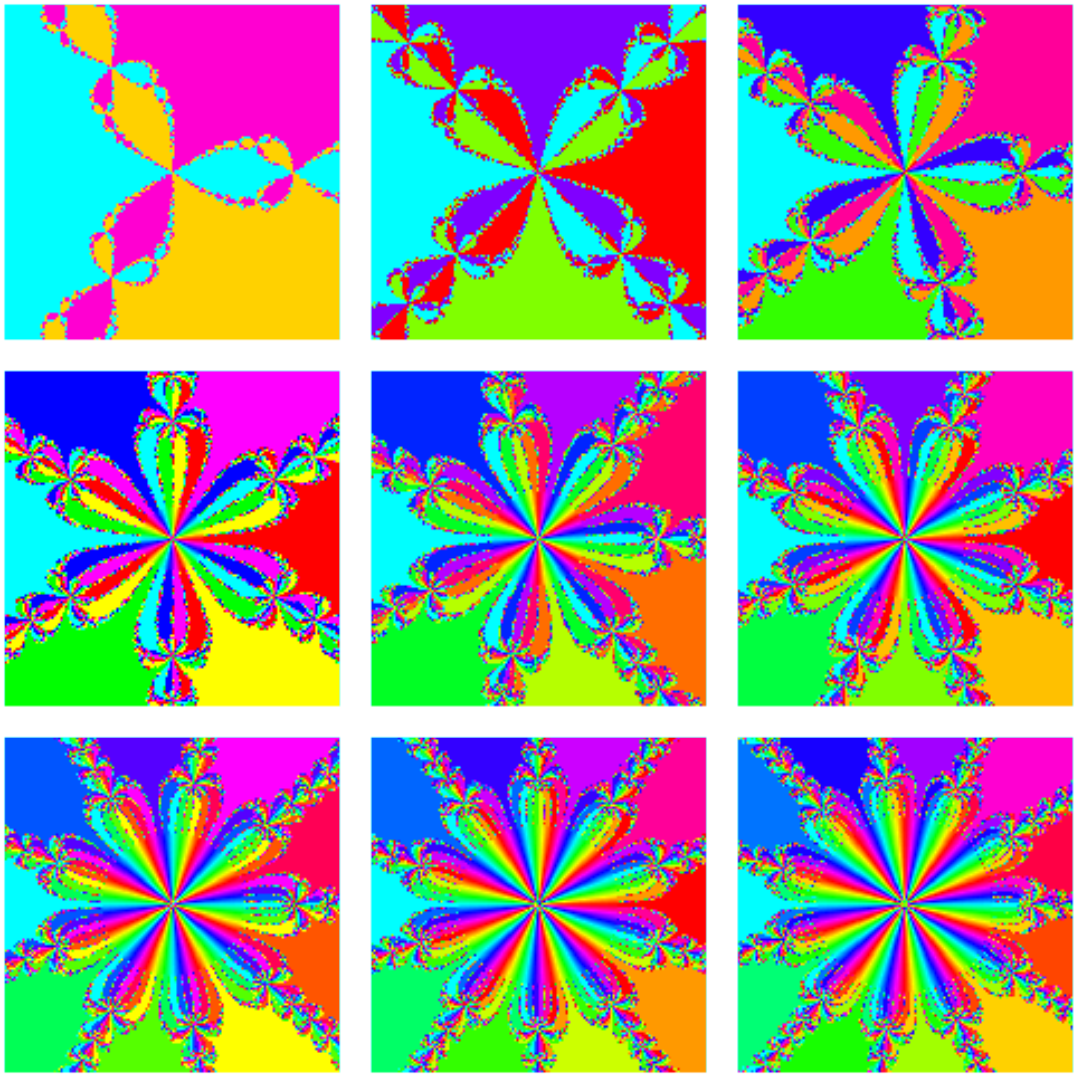
Much harder. Root bracketing cannot be done. The only method that generalizes straightforwardly is Newton-Raphson:

$$\vec{x}_{n+1} = \vec{x}_n - \hat{J}^{-1}\vec{f}$$

$\hat{J} = \left(\frac{\partial f_i}{\partial x_j}\right)$ is the Jacobian matrix

Calculation and inversion of the Jacobian is costly. There are “quasi-Newton” methods that update the Jacobian gradually, rather than calculating it from scratch at every step. E.g., Broyden method.

Newton-Raphson method generates fractals



$$z^n - 1 = 0$$

n roots in the complex plane

Colour sets of initial points converging to a particular root