Lecture 2

Interpolation
Padé Approximants

Summary lecture 1

- introduction
  - Numerical errors from representation doubles
    - Finite mantissa introduces round-off error
  - machine dependent
    - Machine accuracy, about 16 digits for double, 7 for float
    - Round-off errors
    - Truncation errors :
      - A real function has to be approached with a finite number of terms (e.g. Taylor expansion). Ignoring higher-order terms introduces truncation error
      - Minimize with optimal routines
    - Stability : exercise 1
    - Test for stability when applicable
  - Always try to verify the validity of your code

Exercise 1: golden mean

- Example lecture 1. Numerical stability

$$\phi = \frac{\sqrt{5}-1}{2} \approx 0.61$$

$$\phi^n = \exp(n \ln \phi),$$

$$\text{but also via recurrence relation:}$$

$$\phi^0 = 1, \quad \phi^1 = \phi, \quad \phi^{n+2} = \phi^n - \phi^{n+1}$$

- Naive expectation for error: consecutive powers of phi are of the same order (0.6), so the addition is correct to machine accuracy (least significant bit). Each sum would give a relative error of about 10^{-16} (10^{-7}) for double (float)

Precision of calculation: golden mean

- Close inspection however demonstrates that the error grows exponentially, due to the recurrence relation:

$$\phi_{double} = \phi_{exact} + \epsilon_m \quad (\epsilon_m \approx 10^{-16})$$
$$\phi^2_{double} = 1 - \phi_{double} \approx \phi^2_{exact} - \epsilon_m$$
$$\phi^3_{double} = \phi_{double} - \phi^2_{double} \approx \phi^2_{exact} + 2\epsilon_m$$
$$\phi^n_{double} = \phi^{n-2}_{double} - \phi^{n-1}_{double} \approx \phi^n_{exact} - (-1)^n \epsilon_m F_n$$
$$F_n = F_{n-2} + F_{n-1} \quad (Fibonacci \;\; series, 1,1,2,3,5,8,13,.....)$$
$$F_n \approx \frac{F_{n-1}}{\phi_{exact}} \;\Rightarrow\; \frac{\phi^n_{double} - \phi^n_{exact}}{\phi^n_{exact}} \approx \epsilon_m (-1)^n \phi^{-2n}_{exact}$$

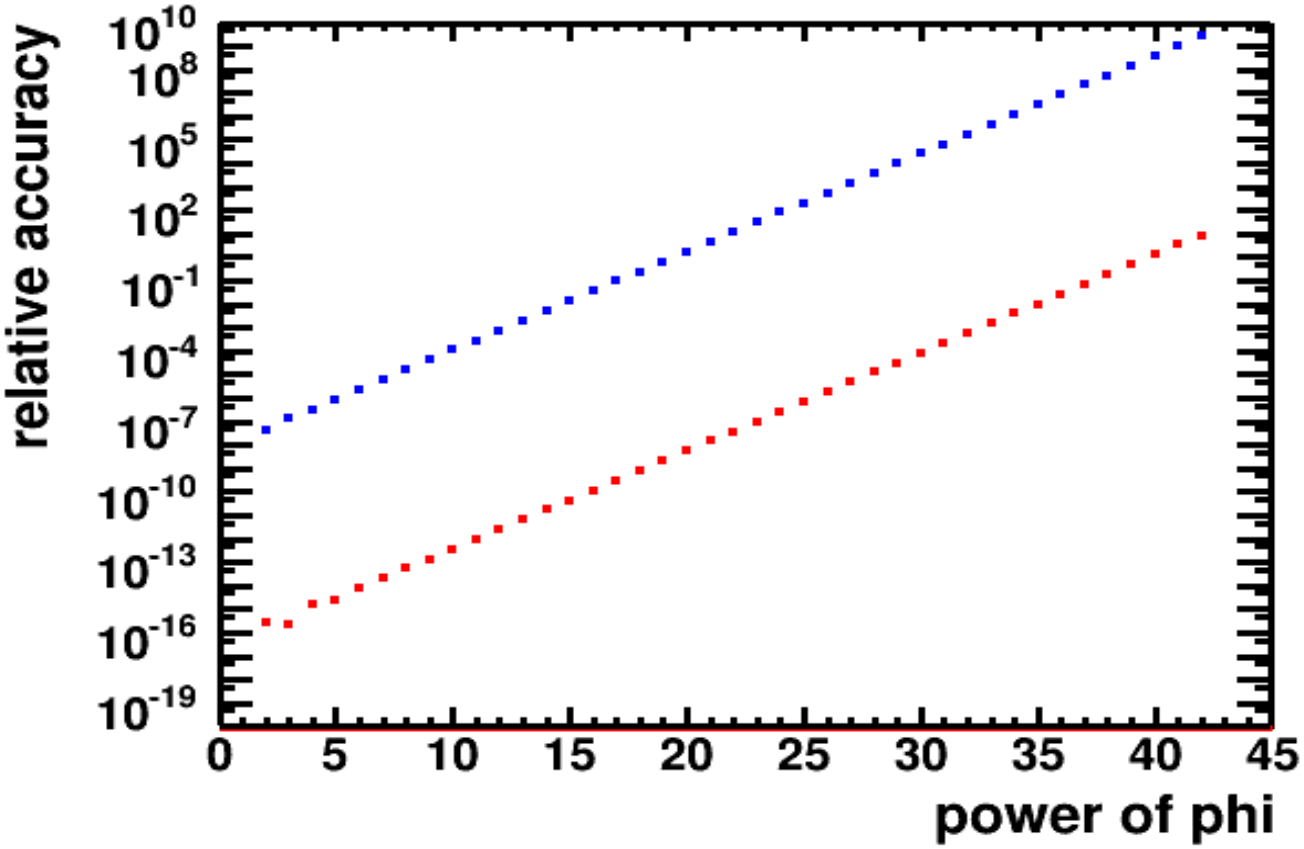
Relative error grows a factor ~ -2.6 per step!

Test for floats/doubles

- code on web
  - Exceeds 0.1% (50 %) after 13 (19) powers of phi for floats
  - Exceeds 0.1% (50 %) after 33 (40) powers of phi for doubles
  - Loss of accuracy : 2 digits every 5 steps!

Note, that the relative accuracy remains at machine precision if one would have calculated phi from the Fibonacci series as

$$1/\phi^n = 1/\phi^{n-1} + 1/\phi^{n-2}$$
$$(1/\phi)^n \approx 1.61\,(1/\phi)^{n-1}$$
$$\epsilon_n \approx 1.61\,\epsilon_{n-1}$$
$$\frac{\epsilon_n}{\phi^n} \approx \epsilon_{machine}$$



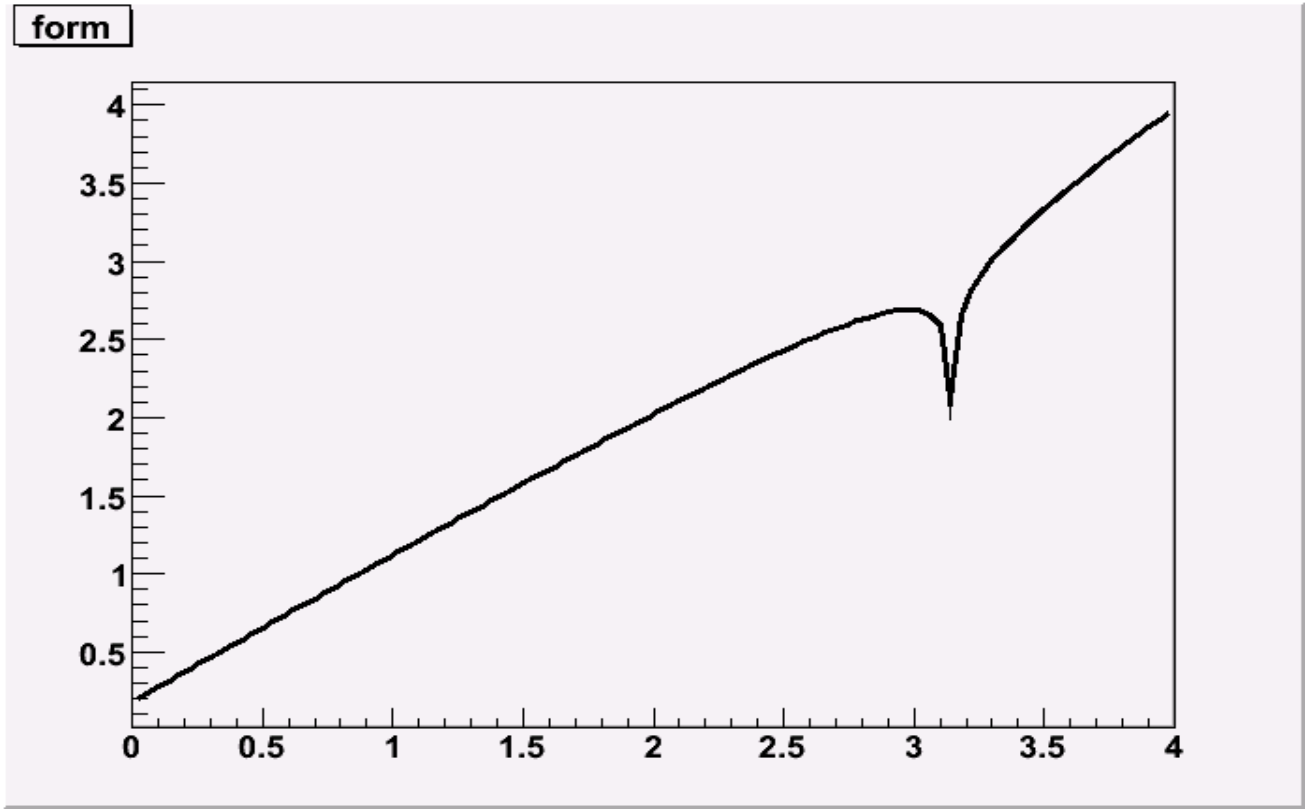
NR chapter 3: Interpolation

- Interpolation of functions
- fixed number of points known
  - from measurements
  - from calculations. e.g continuous-wave Faddeev equation, lattice QCD, ….., takes too much time to calculate per variable value
- calculate points in between
  - error?
- interpolation : in between grid points
- extrapolation -> outside range: dangerous
- However: Padé formalism - analyticity
- interpolation:
  - polynomials
  - trigonometric -> Fourier analysis (later in course)
  - rational functions (Pade)
  - polynomial functions (e.g. Legendre polynomials, Chebyshev polynomials, …)

interpolation

$$x + \frac{\ln|x-\pi|}{2\pi}$$

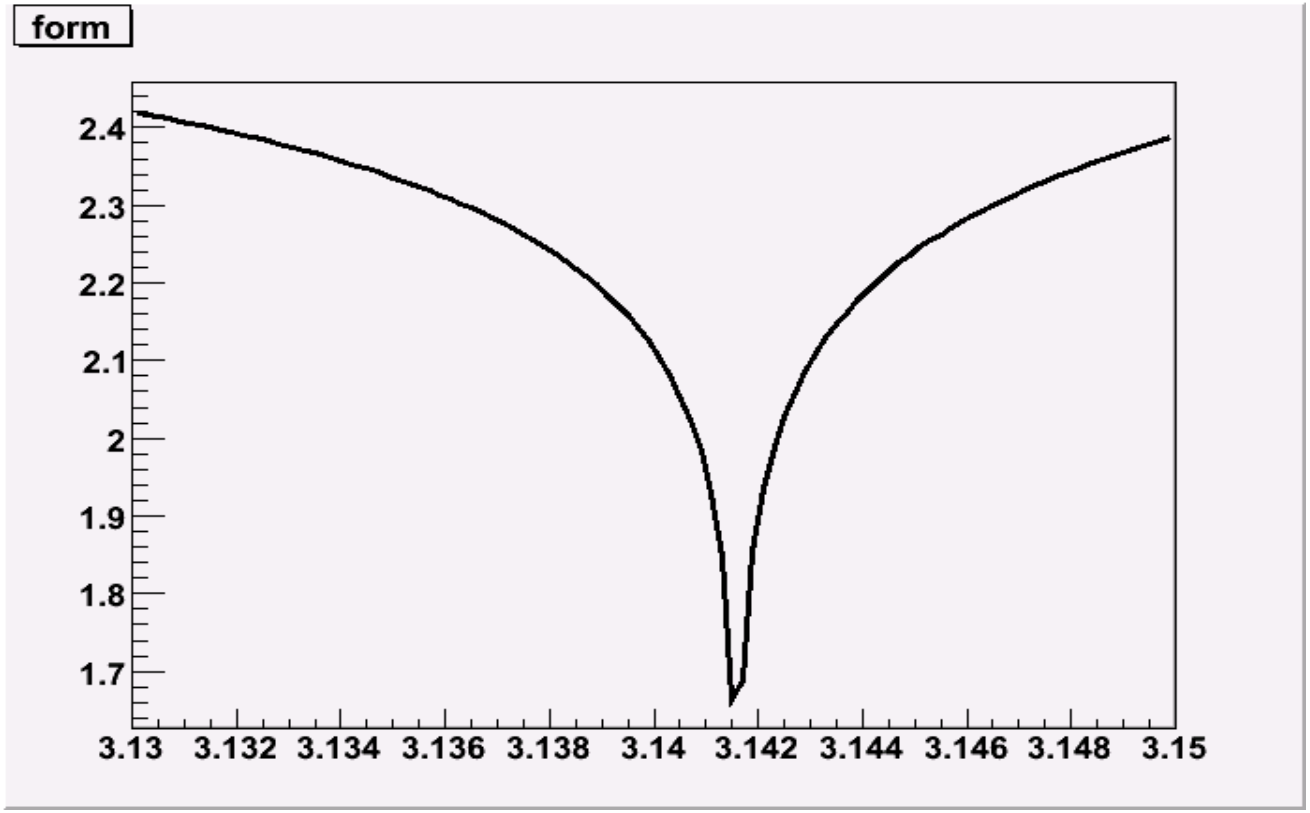
- very weak pole
- mocks all interpolation schemes close to pole



interpolation

$$x + \frac{\ln|x-\pi|}{2\pi}$$

- very weak pole
- mocks all interpolation schemes close to pole



interpolation

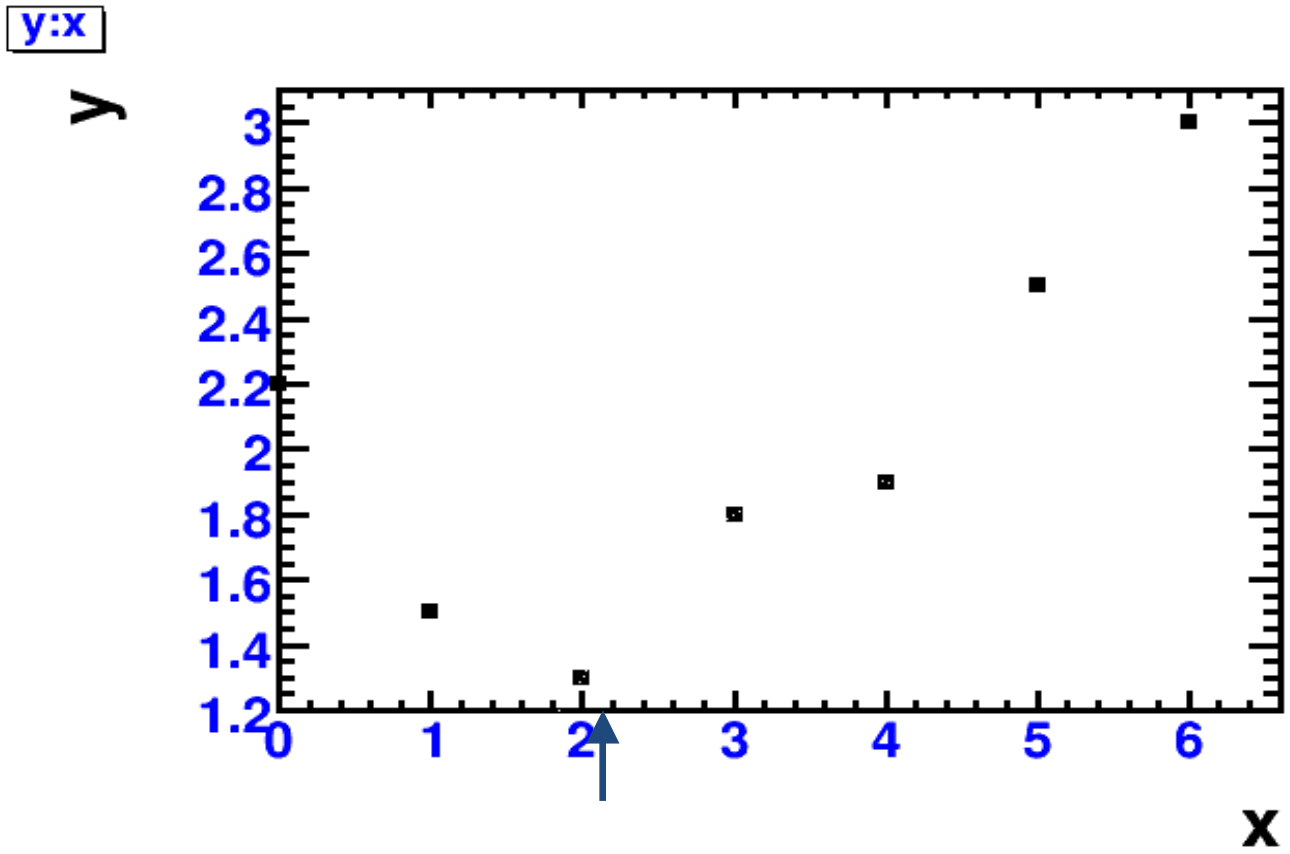
$$x + \frac{\ln|x-\pi|}{2\pi}$$

- very weak pole (<0 for x = pi +/- 3e-9)
- mocks all interpolation schemes close to pole
- Although such a function may go to minus infinity, it can be that there is not any rational number, representable by a double precision variable, that is even negative! If the function values are measured at grid points too far from the pole, an interpolation scheme would not tend to fit the pole correctly

- function approximation:
  - approximate a function by an easier calculable function
  - calculated points of your own choosing! - this freedom in choice of abcissa helps a lot in obtaining accuracy.
  - e.g. Chebyshev function approximation (later this course).
- fitting: functional form is known.
  - no part of the course. may be addressed later when time permits
  - Result deviates from function values at grid points.

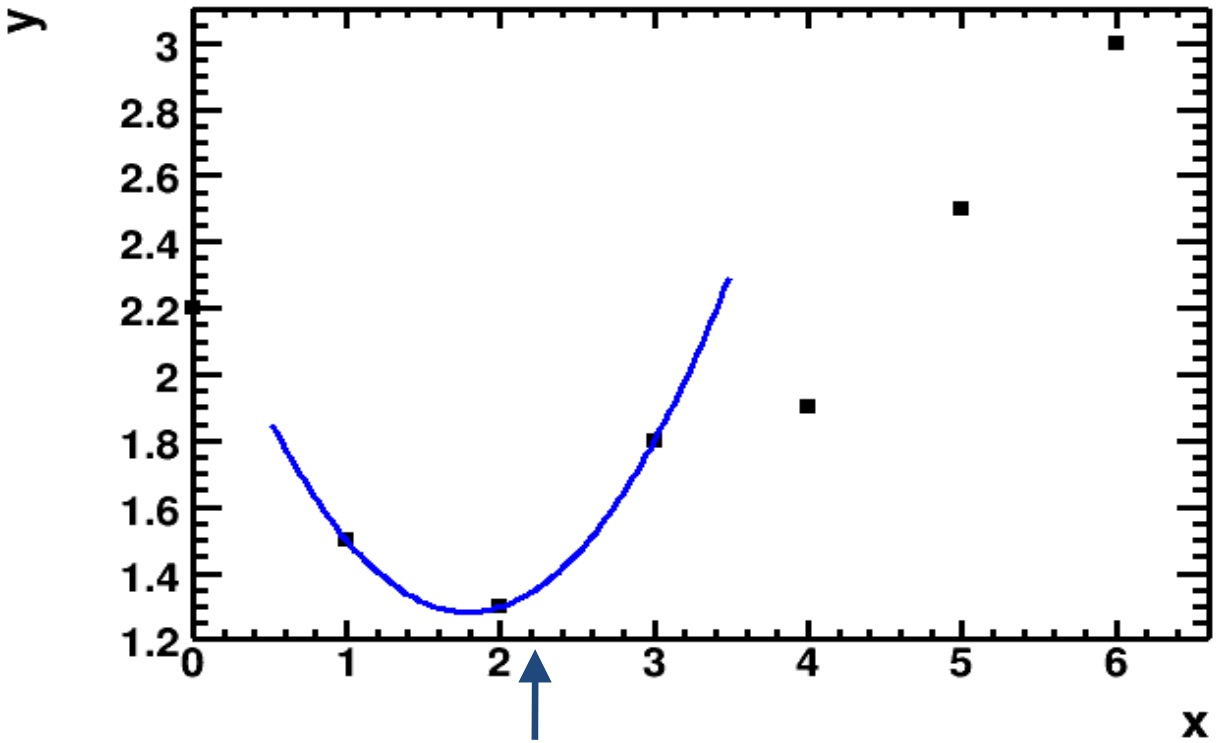
interpolation

- interpolation:
  - use tabulated points around x-value of interest
  - Example: quadratic interpolation (3-points function). Choose 3 points



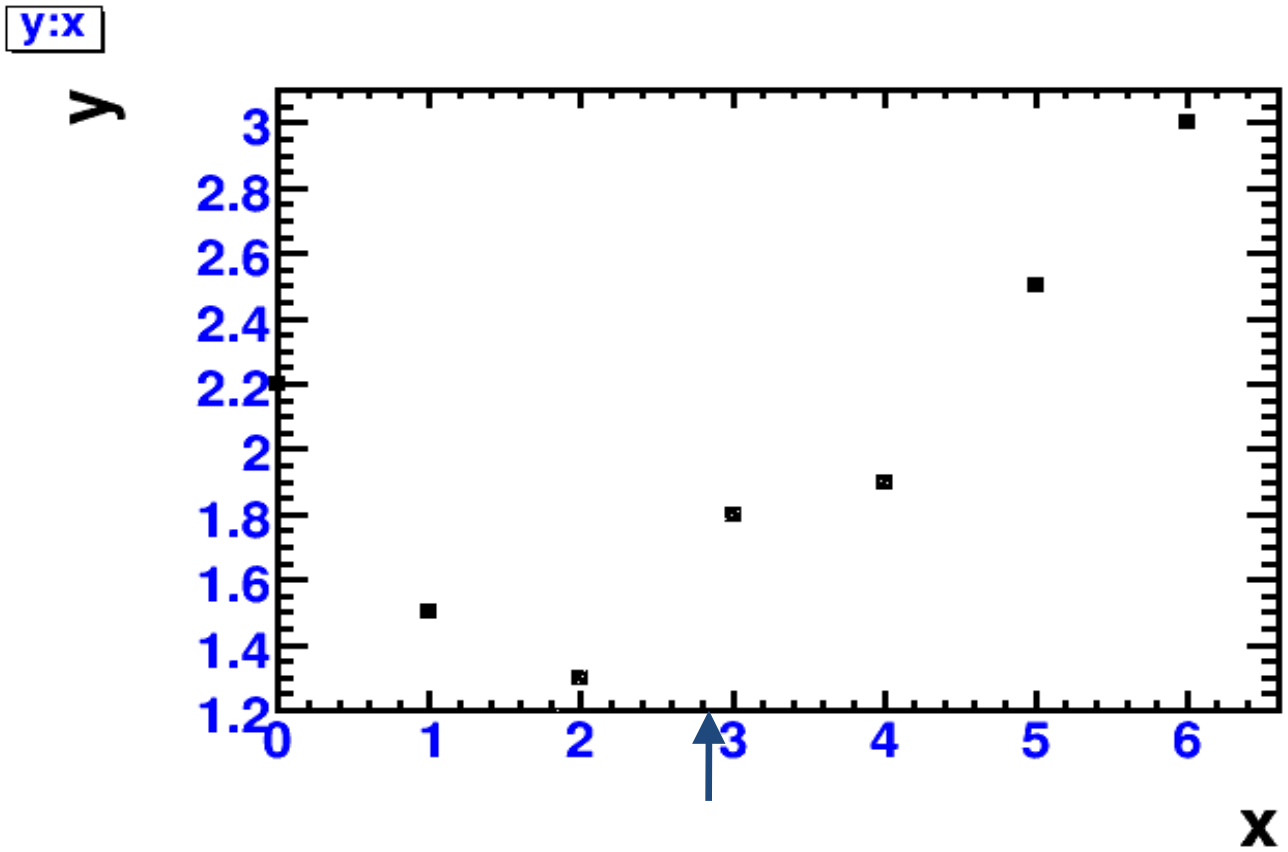
interpolation

- interpolation:
  - use tabulated points around x-value of interest



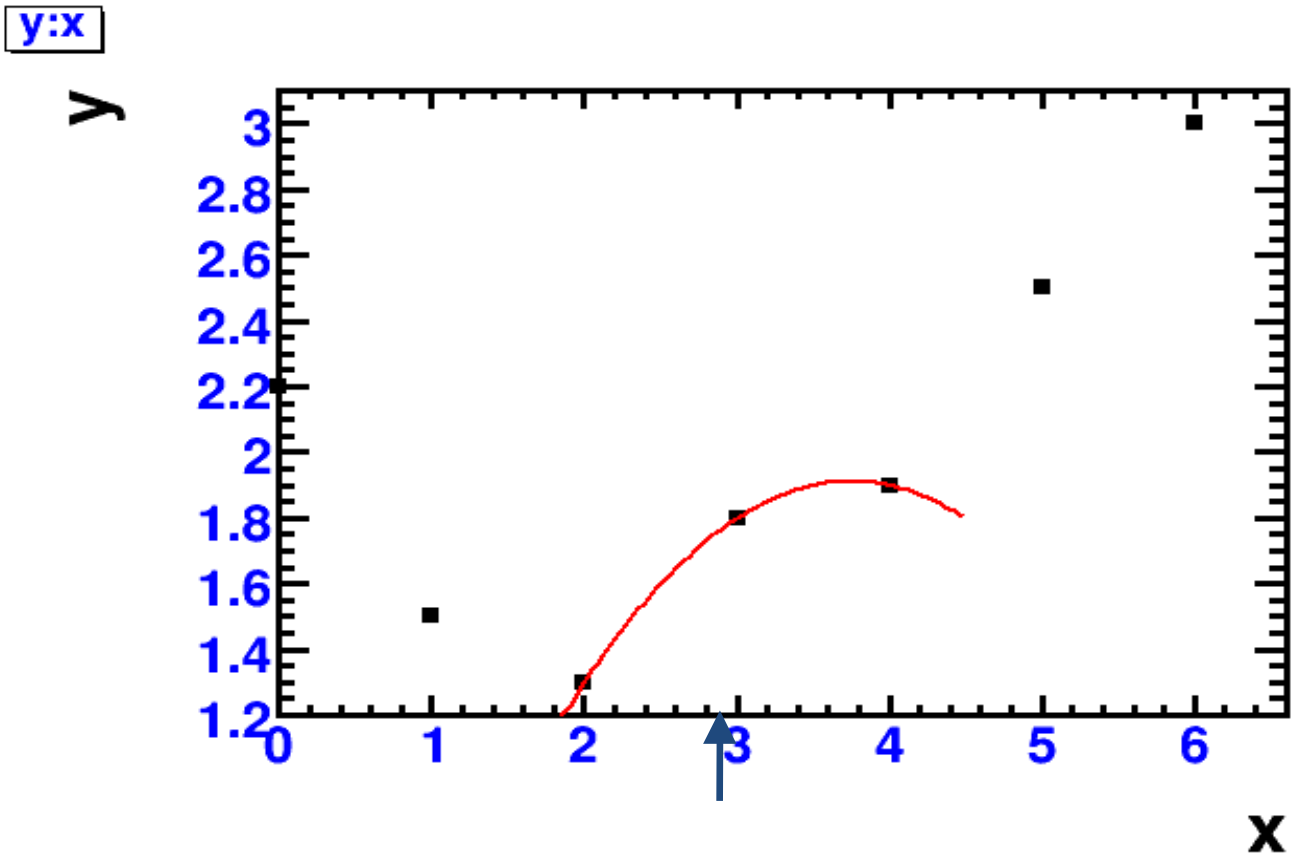
interpolation

- interpolation:
  - use tabulated points around x-value of interest
  - Example: quadratic interpolation (3-points function). Choose 3 points



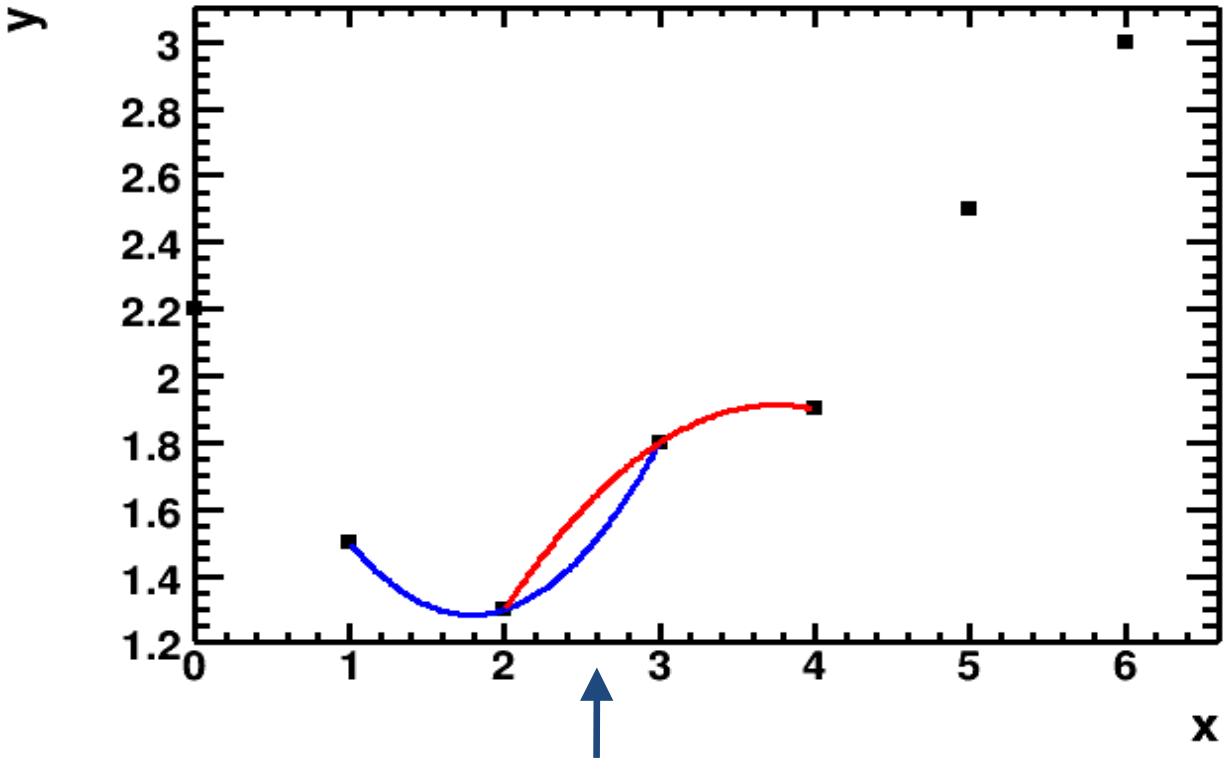
interpolation

- interpolation:
  - use tabulated points around x-value of interest
  - Example: quadratic interpolation (3-points function). Choose 3 points



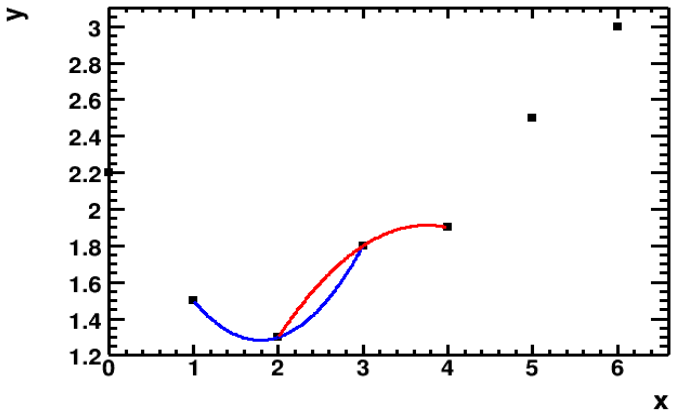
interpolation

- use tabulated points around x-value of interest
- Choose points. Must be consistent! Choosing different points leads to different interpolation, therefore one can only switch choice at a grid point!
- E.g with 3 points, always take two points at lower x and one at higher, or always 1 at lower x and 2 at higher

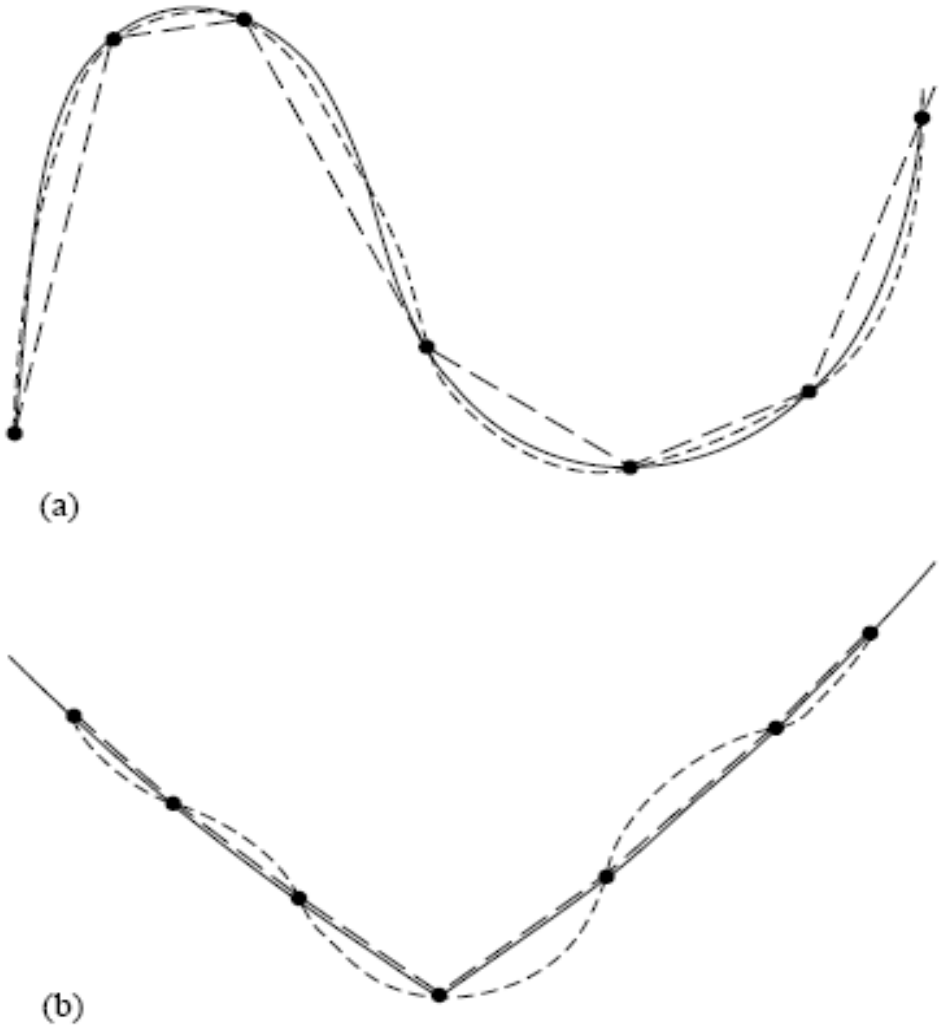


interpolation

- interpolation:
  - use tabulated points around x-value of interest
  - **interpolated function shifts at tabulated point!**
    - Use always same convention for the points to use, change gridpoints in scheme only at $x=x_{grid}$!
  - Continuous function, discontinuous derivatives
  - spline- has smooth derivatives, stiffer function.
  - cubic spline often used (spline)
- higher-order better?



Interpolation



interpolation

- higher order interpolation: fine for smooth functions
- worse for rapidly-changing derivatives (especially polynomial)
- most simple case: polynomial interpolation
- An n-point interpolation uses n grid points. Linear interpolation: 2-point function
- Lagrange polynomials:

$$P(x) = y_1 \frac{x-x_2}{x_1-x_2}\frac{x-x_3}{x_1-x_3}\ldots\frac{x-x_N}{x_1-x_N} +$$

$$+ y_2 \frac{x-x_1}{x_2-x_1}\frac{x-x_3}{x_2-x_3}\ldots\frac{x-x_N}{x_2-x_N} + \ldots. + y_N \frac{x-x_1}{x_N-x_1}\ldots\frac{x-x_{N-1}}{x_N-x_{N-1}}$$

  - goes by construction through n points
  - solved with Neville's algorithm
  - gives error estimate (struct Poly_interp in nr3)

Neville's algorithm

- zeroth order :
- $P_1 = y_1$, $P_2 = y_2$, ... $P_n = y_n$
- higher orders: P_{12}, P_{123}, P_{1234} ..... Polynomials through 12,123,1234
- higher orders recursively obtained:

$$P_{i\ldots i+m} = \{(x - x_{i+m})P_{i\ldots(i+m-1)} + (x_i - x)P_{i+1\ldots i+m}\}/(x_i - x_{i+m})$$

- e.g.

$$P_{12} = \frac{(x - x_2)y_1 + (x_1 - x)y_2}{(x_1 - x_2)}$$

- keep track of difference between $P_{i..i+m}$ and the lower order. This gives error estimate.

$$P_{i..m} - P_{i+1\ldots m} = (x - x_{i+m})/(x_i - x_{i+m})(P_{i\ldots m-1} - P_{i+1..m}) = C_{i,m}$$

$$P_{i..m} - P_{i\ldots m-1} = (x_i - x)/(x_i - x_{i+m})(P_{i+1\ldots m} - P_{i..m-1}) = D_{i,m}$$

Neville's algorithm

$D_{i,m+1} = (x_{i+m+1}-x)(C_{i+1,m} - D_{i,m})/(x_i - x_{i+m+1})$

$C_{i,m+1} = (x_i -x)(C_{i+1,m}-D_{i,m})/(x_i - x_{i+m+1})$

  - error estimate: last difference added. Very useful to have.
  - differences with previous order remembered – recursive formula. Each additional order increases computation time with a linear amount.
- higher-order interpolation can be easily recursively updated
- Note: Poly_interp develops polynomial around the requested x-value
  - Coefficients are minimized for the requested x!
  - Coefficients are re-calculated for each value of x.

interpolation

- **Schemes that do not develop coefficients around the interpolated point will lack in accuracy.** Consider:

$$y = (x - 99.8)$$

$$f(y) = y^{10}$$

$$Accuracy: \quad \sim 10^{-15} f(y)$$

$$f(x) = \sum_{i=0}^{10} \binom{10}{i} x^i (-99.8)^{10-i}$$

$$Accuracy :\approx 10^{-15}(x^{10} + ... + (-99.8)^{10}) <\approx 10^5 \qquad (x \sim 0 - 100)$$

- Therefore: use well-tested library algorithms for interpolation (except for linear interpolation, that is easy to code yourself)

Numerical Recipes routines

- Include “nr3.h”

http://www.nikhef.nl/~henkjan/NUMREC/include.zip

  - Doub, VecDoub, nrerror, etc.
  - Using namespace std, iostream, etc

    In source code:

    VecDoub xx(n); creates a vector of doubles

    xx can be used as a C array, e.g. xx[i]=xx[i-1]+3;

    VecDoub is not a std::<vector>, but the method .size() is supported.

- Include interp_1d.h
  - 1-dimensional interpolation, used in the other interpolation routines.
  - Hunting for grid points to use. Assigning the error estimate, and the interpolated value. Implements methods for Polynomial interpolation, rational interpolation, cubic splines

    usage e.g:

    VecDoub xx(100),yy(100); ….. fill xx and yy with the abscissa and function values…..

    Poly_interp pol(xx,yy,n); -> creates a Poly_interp object named pol, using the vectors xx and yy. Will perform n-point polynomial interpolation. For n=2: linear interpolation.

    For n=4 : 3rd order polynolmial ($c0 + c1*x + c2*x^2+c3*x^3$)

    double r=17.3; double y=pol.interp(r); double err=pol.dy;

    Returns interpolated result for r=17.3 in y, and error estimate in err.

Example include file, how it works

Open the include file with an editor or C++

/home/henkjan/NR3/interp_1d.h

```
struct Base_interp
{
        Int n, mm, jsav, cor, dj;
        const Doub *xx, *yy;
        Base_interp(VecDoub_I &x, const Doub *y, Int m)
                : n(x.size()), mm(m), jsav(0), cor(0), xx(&x[0]), yy(y) {
                dj = MIN(1,(int)pow((Doub)n,0.25));
        }

        Doub interp(Doub x) {
                Int jlo = cor ? hunt(x) : locate(x);
                return rawinterp(jlo,x);
        }

        Int locate(const Doub x);
        Int hunt(const Doub x);

        Doub virtual rawinterp(Int jlo, Doub x) = 0;

};

    struct Poly_interp : Base_interp
    {
            Doub dy;
            Poly_interp(VecDoub_I &xv, VecDoub_I &yv, Int m)
                    : Base_interp(xv,&yv[0],m), dy(0.) {}
            Doub rawinterp(Int jl, Doub x);
    };
```

Base class with general methods, and the data

Constructor

Function call to get interpolated result

Virtual function, must be implemented in derived class

Xv,yv: vectors of grid points (x,y).
m: m-point function. Linear interpolation: m=2

Rational interpolation

- -poles
- (complex plane – ruins convergence)
- rational functions will stay good, as long as there are enough powers in denominator
- Pade approximation, chapter 5.12
- R(x) = P(x)/Q(x):
  - by definition $Q_0 = 1$
  - n-th order goes through n+1 points: mu+nu+1=n
  - Bulirsch and Stoer : Neville-type algorithm
    - diagonal: mu (+1) = nu, as many powers in P(x) as in Q(x)
    - also gives error estimate by comparison with previous order
  - In NR version 3: Rat_interp and BaryRat_interp

Padé extrapolation

- A Padé approximant is the rational function that has a power series expansion that agrees to a given power series to the highest known order M+N (with M=N or M+1=N)

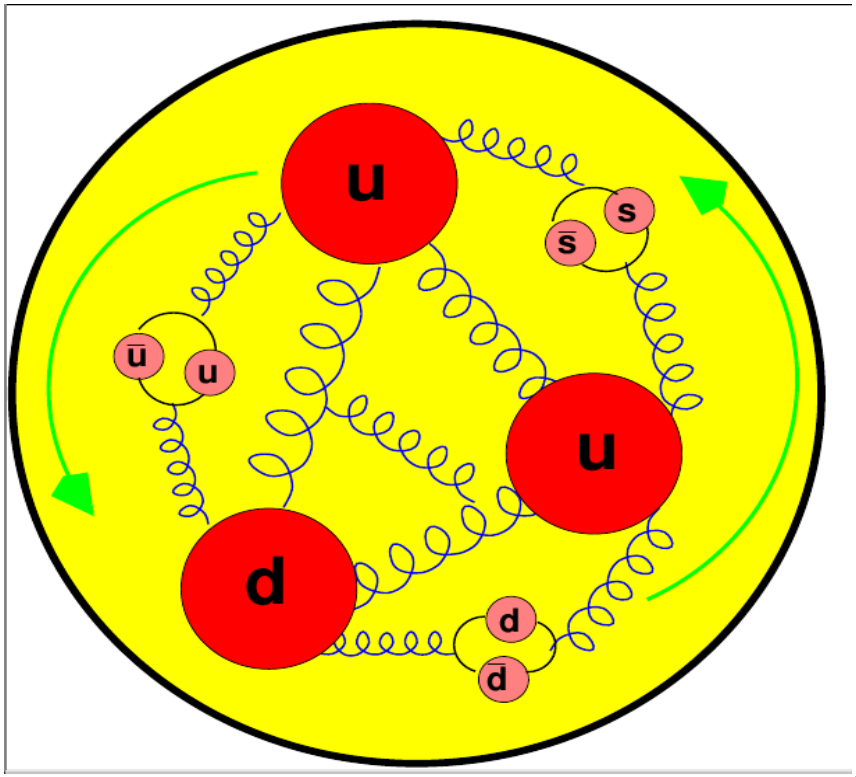
$$R(x)=\frac{\sum_{k=0}^{M} a_k x^k}{1+\sum_{k=1}^{N} b_k x^k} \quad , \quad f(x)=\sum_{k=0}^{\infty} c_k x^k$$

$$R(0)=f(0) \quad \text{and} \quad \left[\frac{d^k}{dx^k} R(x)\right]_{x=0} = \left[\frac{d^k}{dx^k} f(x)\right]_{x=0} \qquad (k=0....M+N)$$

- very useful if you can calculate a function and several orders of the derivative.
- Widely used in literature (e.g. > 1000 hep articles have Pade in the abstract/title)

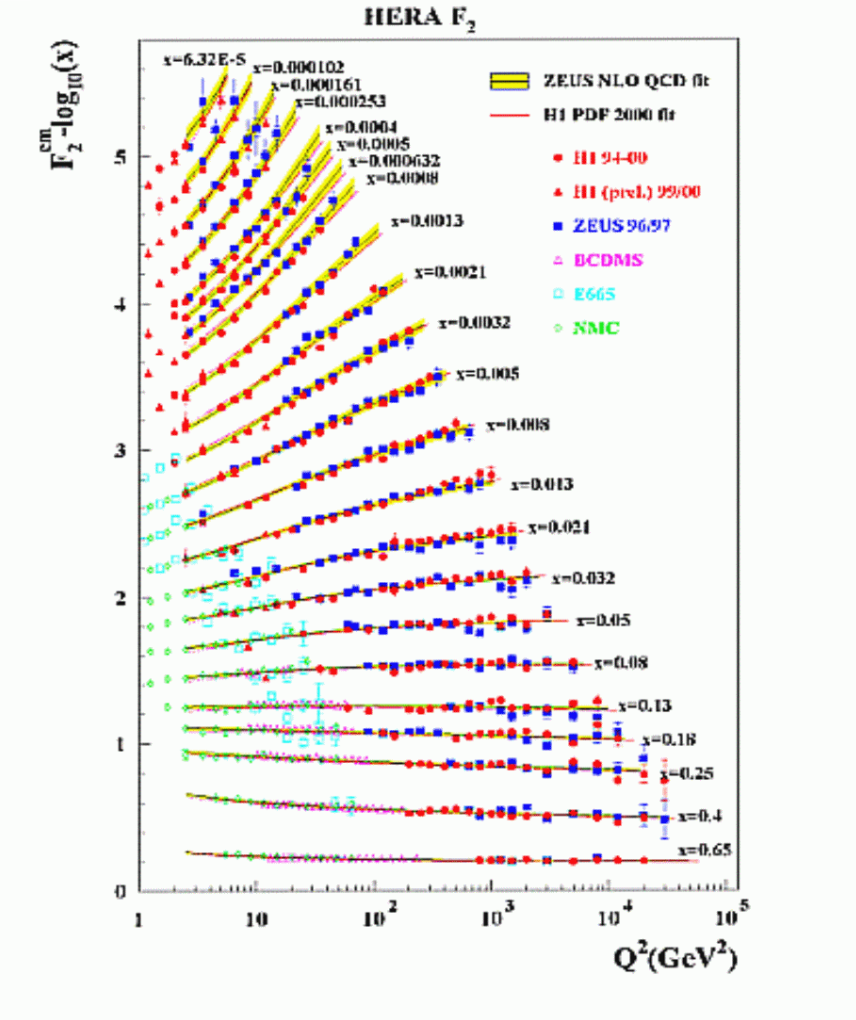
Pade Approximation, example

- Deep-inelastic scattering: structure functions are determined and interpreted in terms of quark/gluon contents in the nucleon. (Spin and momentum of the nucleon)

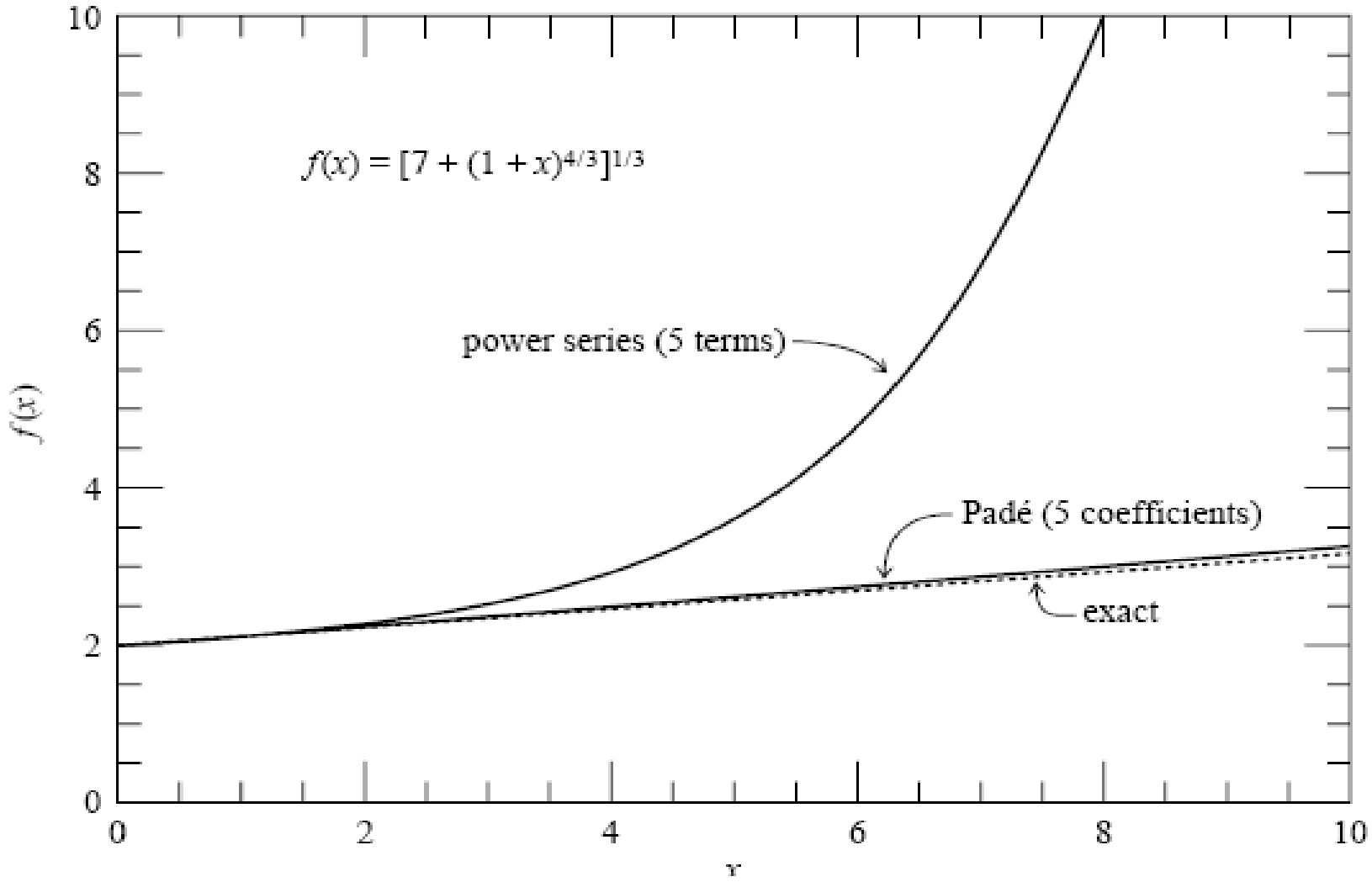


Pade Approximation, example

- Deep-inelastic scattering: structure functions are determined and interpreted in terms of quark/gluon contents in the nucleon. (Spin and momentum of the nucleon)
- more sea-quarks resolved at higher momentum transfer.
- structure functions can be evolved from one value of x,Q^2 to another Q^2 via DGLAP, third order QCD calculations. Pade extrapolation is used.
- DGLAP evolutions, Dokshitzer-Gribov-Lipatov-Altarelli-Parisi https://de.wikipedia.org/wiki/DGLAP-Gleichungen
  Guido Altarelli (2009), Scholarpedia, 4(1):7124
-
- Also: see e.g http://www.mdpi.com/2075-4434/4/1/4 for an example in cosmological expansion.



Pade approximation, ch. 5.12



Pade approximation

- Pade.h : needs poly.h, ludcm.h, nr3.h. contains Ratfn pade(VecDoub_I &coef) : call pade with as argument a NR vector (VecDoub coef), containing the coefficients of the power series approximation of the function you want to approximate.
- Return value: a Ratfn. This Ratfn is an object with a constructor Ratfn(VecDoub, VecDoub) that generates the interpolation function. The () operator is overloaded. Calling Ratfn(Doub) returns the Pade approximation.
- Usage: VecDoub coef(N); // make vector with N coefs
- Ratfn rat=pade(coef); // make rat = Ratfn function
- Double x = 0.4; double y=rat(x); // calculate y = pade approximation for x=0.4.

```
Ratfn pade(VecDoub_I &cof)
{
	Int j,k,n=(cof.size()-1)/2;
	Doub sum;
	MatDoub q(n,n),qlu(n,n);
	VecDoub x(n),y(n),num(n+1),denom(n+1);
	for (j=0;j<n;j++) {
		y[j]=cof[n+j+1];
		for (k=0;k<n;k++) q[j][k]=cof[j-k+n];
	}
	LUdcmp lu(q);
	lu.solve(y,x);
	for (j=0;j<4;j++) lu.mprove(y,x);
	for (k=0;k<n;k++) {
		for (sum=cof[k+1],j=0;j<=k;j++) sum -= x[j]*cof[k-j];
		y[k]=sum;
	}
	num[0] = cof[0];
	denom[0] = 1.;
	for (j=0;j<n;j++) {
		num[j+1]=y[j];
		denom[j+1] = -x[j];
	}
	return Ratfn(num,denom);
}
```

```
struct Ratfn {
	VecDoub cofs;
	Int nn,dd;

	Ratfn(VecDoub_I &num, VecDoub_I &den) : cofs(num.size()
+den.size()-1),
	nn(num.size()), dd(den.size()) {
		Int j;
		for (j=0;j<nn;j++) cofs[j] = num[j]/den[0];
		for (j=1;j<dd;j++) cofs[j+nn-1] = den[j]/den[0];
	}

	Ratfn(VecDoub_I &coffs, const Int n, const Int d) : cofs(coffs), nn(n),
	dd(d) {}

	Doub operator() (Doub x) const {
		Int j;
		Doub sumn = 0., sumd = 0.;
		for (j=nn-1;j>=0;j--) sumn = sumn*x + cofs[j];
		for (j=nn+dd-2;j>=nn;j--) sumd = sumd*x + cofs[j];
		return sumn/(1.0+x*sumd);
	}

};
```

Spline

linear interpolation:

$$y = Ay_j + By_{j+1}$$

$$A = (x_{j+1} - x)/(x_{j+1} - x_j)$$

$$B = (x - x_j)/(x_{j+1} - x_j)$$

zero second derivative, discontinuous first on grid points

Cubic spline: continuous on second derivative, smooth on first

How to proceed?: add 3rd-order polynomial, such that y'' linearly varies from y''_i to y''_{i+1}.

Polynomial constructed to be zero at grid points

$$y = A\,y_j + B\,y_{j+1} + C\,y''_j + D\,y''_{j+1}$$

$$C = \frac{1}{6}\left(A^3 - A\right)\left(x_{j+1} - x_j\right)^2$$

$$D = \frac{1}{6}\left(B^3 - B\right)\left(x_{j+1} - x_j\right)^2$$

Spline

- This yields for the 2nd derivative:

$$\frac{dy}{dx} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} + \frac{\left(3B^2 - 1\right) y''_{j+1} - \left(3A^2 - 1\right) y''_j}{6} \left(x_{j+1} - x_j\right)$$

$$\frac{d^2 y}{dx^2} = A\, y''_j + B\, y''_{j+1}$$

- second derivative linearly changes between x_j and x_{j+1}
- However: y'' not known.
- Require that first derivative y' is continuous across boundary
  - use this to get y''

- equate y'_j from j-1,j and j,j+1
- this yields for j=1 .. N-2 :

$$\left(x_j - x_{j-1}\right) y''_{j-1}/6 + \left(x_{j+1} - x_{j-1}\right) y''_j/3 + \left(x_{j+1} - x_j\right) y''_{j+1}/6 = = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}$$

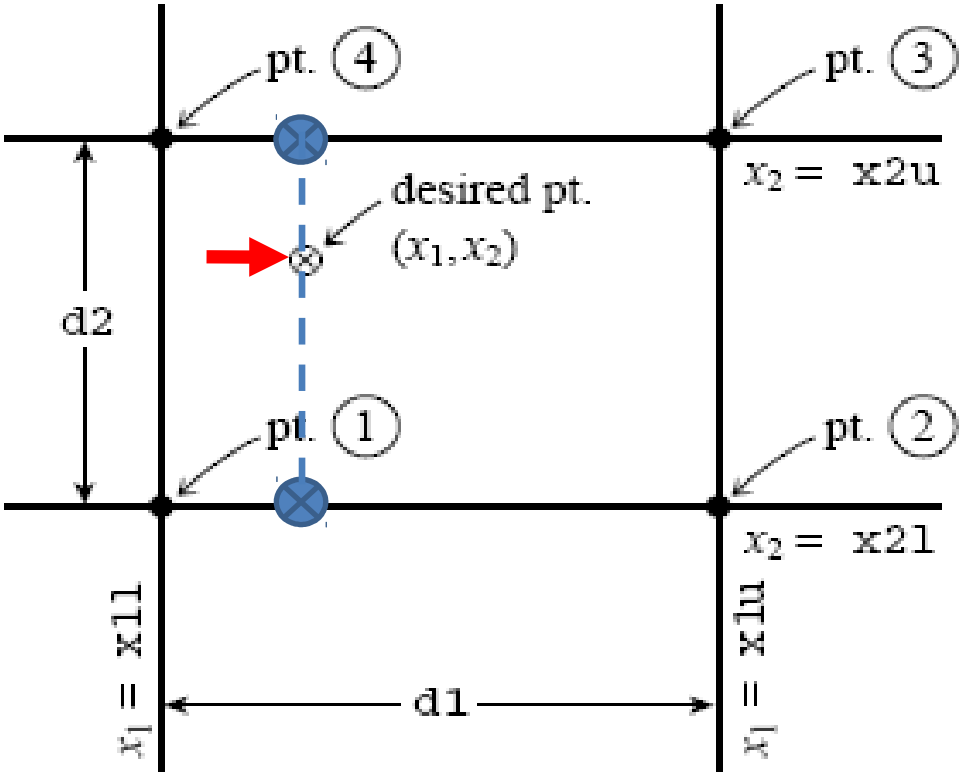
Spline

- N-2 equations, N unknowns
- set y''_0 and y''_{N-1} zero (linear extrapolation outside boundaries)
  - **this is called natural cubic spline**
- or set either y''(0) and y"(N-1) to values to give a specified value of y'(0) and/or y'(N-1)
- cubic spline: tri-diagonal set of equations. Each y'' is coupled to its two neighbors.
- O(N) calculations to solve the N equations.
  - for general schemes one would expect $O(N^3)$ calculations
- NR ed. 1,2; spline: call only once. Store parameters. Splint routine gives interpolated values. NR ed. 3: Spline_interp.
- spline typically available in operating system (e.g. linux)

Interpolation in multiple dimensions

- much harder to obtain good interpolation
- linear interpolation in two directions often done.
- bi-cubic splines is an alternative. Smooth derivatives
- take out factors that you know! E.g. if a function f(a,b) has a factor exp(a), divide it out.
- If derivatives can be calculated, higher accuracy may be obtained (also for bi-cubic spline).

interpolation in 2 dimensions



(a)

pt. number

	1	2	3	4
y				
$\partial y/\partial x_1$		user supplies these values		
$\partial y/\partial x_2$				
$\partial^2 y/\partial x_1 \partial x_2$				

(b)

Exercise 3

Exercise 3. Due Friday, Feb. 15, 2017 (12:00). weight 2 points.

Assume, that the values of the function $y(x)=\ln(|x+\pi|)$ are known for the values of $x=-20+0.25\,i$, with 81 integers i from 0 to 80. Interpolate these function values for all values of $x=-10+0.01\,j$, from x=-10 to x=+10, using the natural cubic spline, a (N-1)-order polynomial (i.e. a N-point interpolation function), and the power series and Pade approximant for N coefficients.

For this, assume that at x=0 the first N-1 derivatives of $f(x)$ are known. The power series can be written as $\ln(|x+\pi|)=\sum^{N-1}\frac{1}{\ldots}x^{\ldots}$ for values of $x<-\pi$ (Check). For $x>-\pi$ one can replace the x in the power series by $x+2\pi$ or alternatively create a new power series, that takes into account that the absolute value of $x=-2\pi+x$ equals -($x+\pi$) for $x<-\pi$, which is more work (you need 2 functions and 2 Pade approximants in that case). The coefficients of the power series can be used to calculate the function value at all values of x and also to calculate the Pade approximant at all values of x.

Determine the average deviation from the true function result to the interpolated result, give the RMS value of the deviations of the interpolated results for each of these interpolation schemes (i.e. calculate $\sqrt{\ldots}$) for the 2001 values of x for the 2 interpolation schemes, the power series, and the Pade approximation. Do this for N=4,7,10,13 and 16 (so interpolation and approximation with 4,7,10,13 and 16 coefficients).

Note, that the power series and Pade approximants extrapolate results over the full range in x, while the interpolated results need 81 function determinations and interpolate between grid points around the value of x. Therefore, it is surprising how accurate the Pade approximation turns out to be.