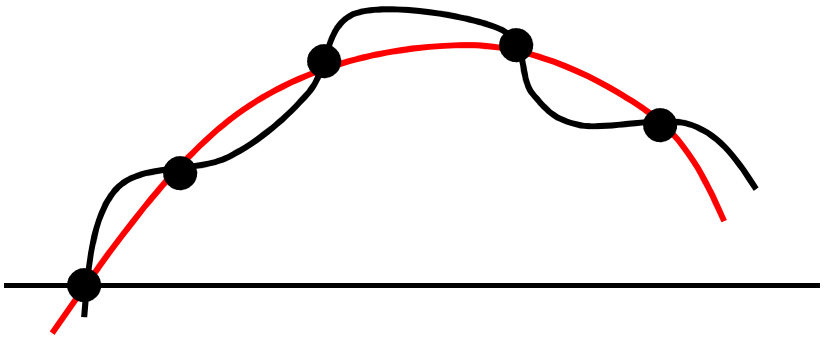
Lecture 6

- Spline interpolation
- Evaluation of functions: series.
- Evaluation of functions: polynomials, complex numbers.
- Recurrence relations

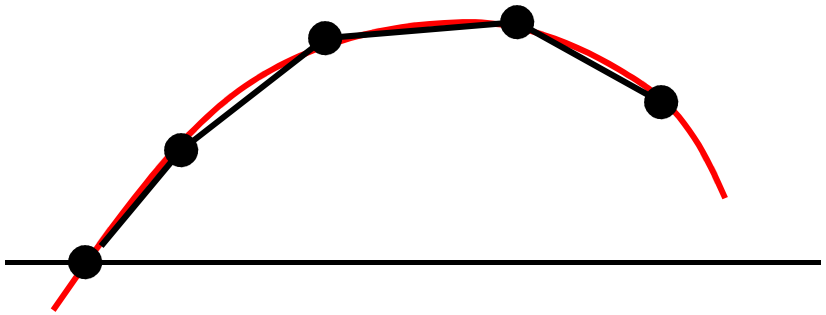
Spline interpolation:

We argued that polynomial interpolations of high degree tend to generate unwanted oscillations.



A way to avoid these oscillations is to subdivide the set of points that one wishes to interpolate and fit each subdivided set with a polynomial of low degree.

For instance, one can take pairs of points, and fit straight lines.



The resulting function "looks better", but has zero second derivatives in the interior and ill-defined second derivatives at the points which one interpolates.

One could try the following: pick pairs of points and fit quadratics. Then one has a free parameter per pair of points. Choose such parameters in such a way to ensure that the function has a continuous first derivative.

Since in physics we are normally interested in computing things up to second derivatives, it is more natural to consider cubic interpolants. That way we have four unknowns per pair of points and we can guarantee that the first and second derivatives will exist and are continuous. Such approximation is called cubic spline interpolation and it is quite common.

A cubic spline interpolant of a given function f is a function with the following properties,

- S(x) is a cubic polynomial, denoted $S_j(x)$ in the subinterval $[x_j,x_{j+1}]$ for each j=0,1,…n-1.
- $S_j(x_j)=f(x_j)$ for each j=0,1,…n
- $S_{j+1}(x_{j+1})=S_j(x_{j+1})$ for each j=0,1,…n-2
- $S'_{j+1}(x_{j+1})=S'_j(x_{j+1})$ for each j=0,1,…n-2
- $S''_{j+1}(x_{j+1})=S''_j(x_{j+1})$ for each j=0,1,…n-2
- One of the following set of boundary conditions are satisfied,
  I) $S''(x_0)=S''(x_n)=0$ (free or "natural" boundary)
  II) $S'(x_0)=f'(x_0)$ and $S'(x_n)=f'(x_n)$ ("clamped" boundary)

So let us construct one such polynomial

- S(x) is a cubic polynomial, denoted $S_j(x)$ in the subinterval $[x_j,x_{j+1}]$ for each j=0,1,…n-1.
- $S_j(x_j)=f(x_j)$ for each j=0,1,…n
- $S_{j+1}(x_{j+1})=S_j(x_{j+1})$ for each j=0,1,…n-2
- $S'_{j+1}(x_{j+1})=S'_j(x_{j+1})$ for each j=0,1,…n-2
- $S''_{j+1}(x_{j+1})=S''_j(x_{j+1})$ for each j=0,1,…n-2
- One of the following set of boundary conditions are satisfied,
  I) $S''(x_0)=S''(x_n)=0$ (free or "natural" boundary)
  II) $S'(x_0)=f'(x_0)$ and $S'(x_n)=f'(x_n)$ ("clamped" boundary)

$$S_j(x) = a_j + b_j(x-x_j) + c_j(x-x_j)^2 + d_j(x-x_j)^3$$

The second condition implies $S_j(x_j) = a_j = f(x_j)$

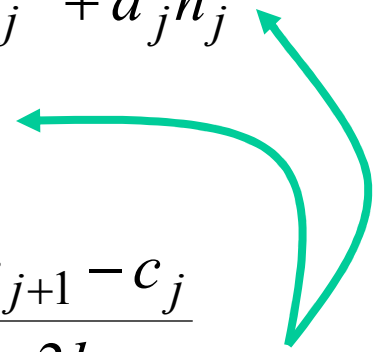
The third condition implies

$$S_{j+1}(x_{j+1}) = a_{j+1} = a_j + b_j(x_{j+1}-x_j) + c_j(x_{j+1}-x_j)^2 + d_j(x_{j+1}-x_j)^3$$

Notation : $h_j = x_{j+1} - x_j$ $\qquad a_{j+1} = a_j + b_j h_j + c_j h_j^{\,2} + d_j h_j^{\,3}$

Fourth condition: $\qquad b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^{\,2}$

Fifth condition: $\qquad c_{j+1} = c_j + 3d_j h_j \qquad \Rightarrow d_j = \dfrac{c_{j+1}-c_j}{3h_j}$



First substitution:

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})$$

$$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j)$$

Now substituting in $b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2$

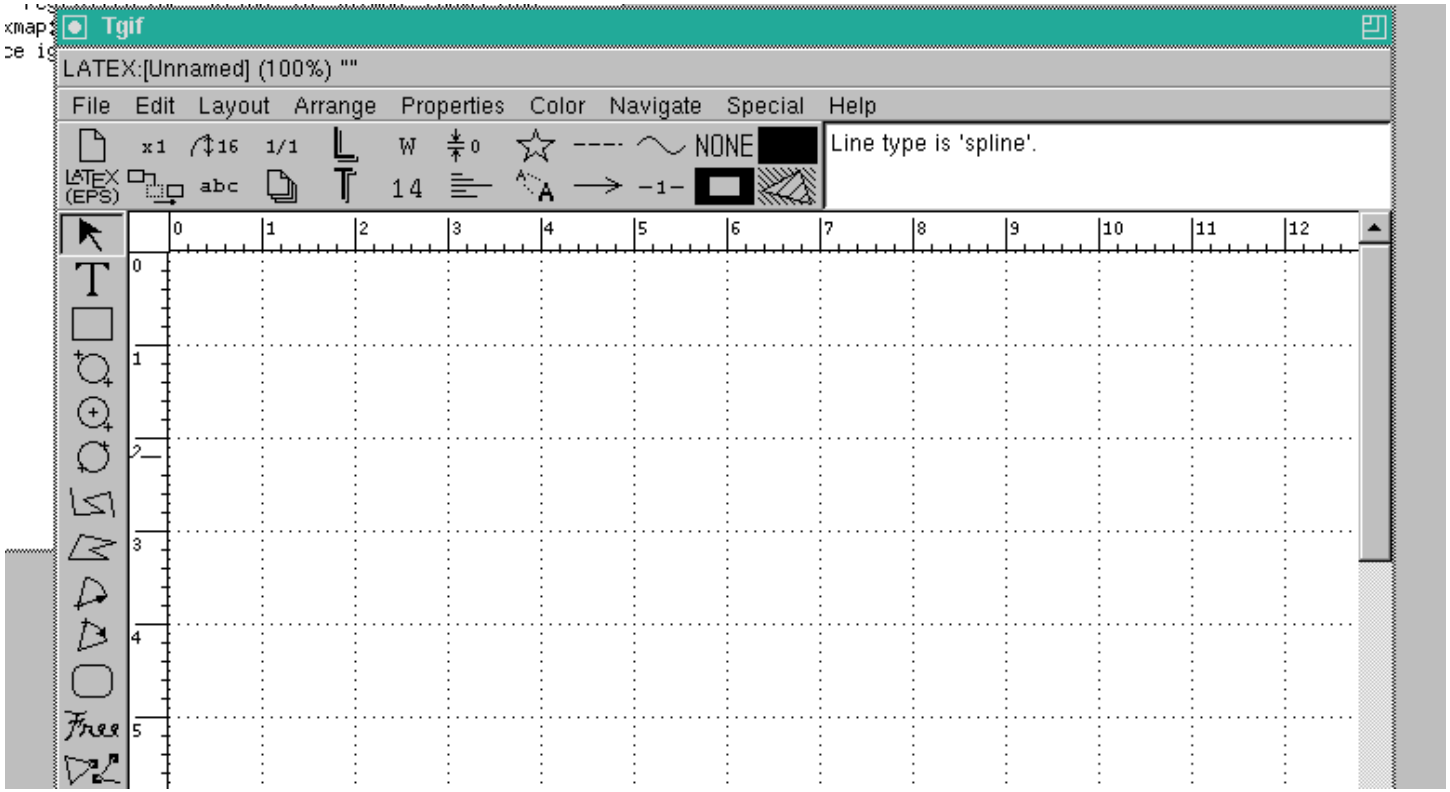
$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1})$$

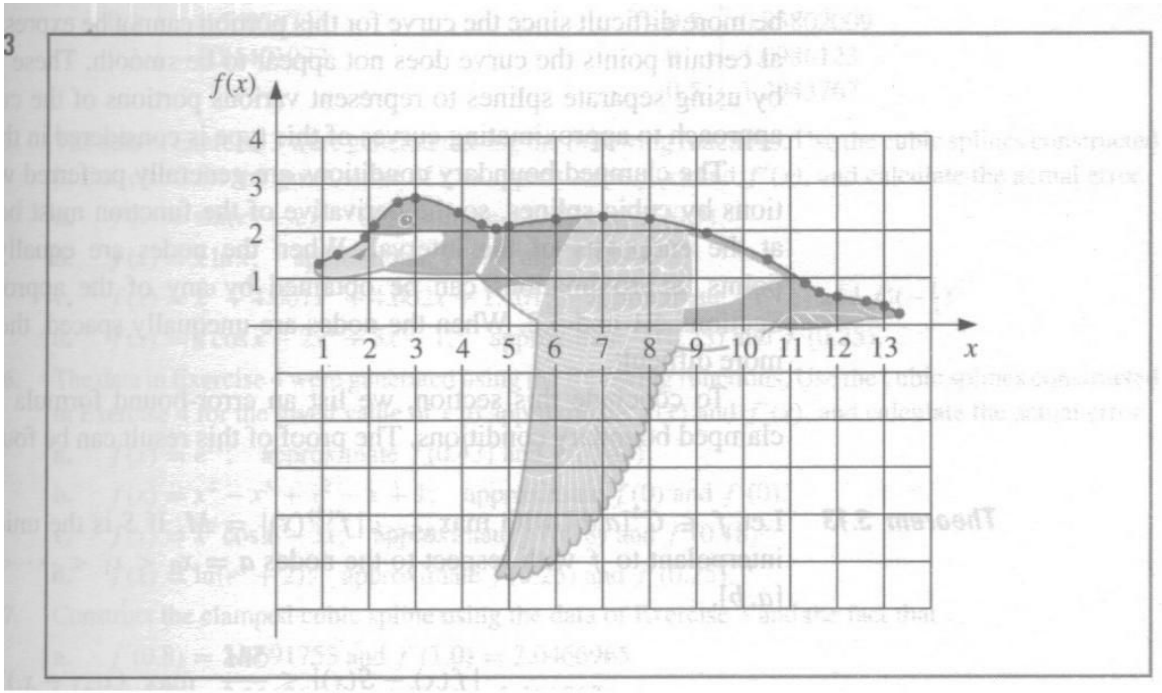
This system involves only the c's as unknowns since the a's we determined in the very first equation and the h's are just the spacings of the points in which we specify the function.

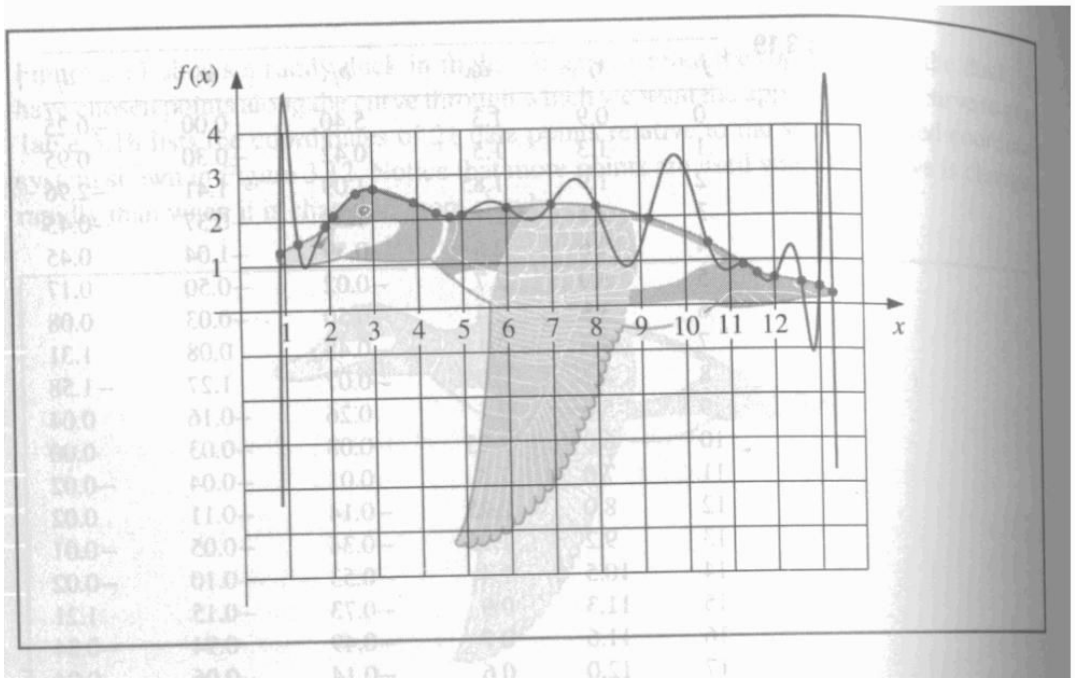
Having the c's, one can then find the b's and d's.

With the boundary conditions specified, this leads to a system of linear equations in which the matrix is “diagonally” dominant (meaning the only non-zero components are the diagonal and “two parallel lines”). Such matrices are non-singular and therefore the problem is solvable (details: Burden & Faires).

Splines are commonly used in computer drawing programs.




f(x)
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13
x


f(x)
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12
x

Osculating polynomials

Given $x_0, x_1, \ldots x_N$ and values of a function f(x), $f_0, f_1, \ldots, f_N$
and integers $m_0, m_1, \ldots, m_N$
the osculating polyomial has that $P(x_k)=f(x_k)$ and all derivatives
up to $P^{(m_k)}(x_k) = f^{(m_k)}(x_k)$

The important case $m_i=1$ is given by the Hermite polynomials:

$f(x) = H_{2n+1}(x)$ with $H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j) H_{n,j}(x) + \sum_{j=0}^{n} f'(x_j) \hat{H}_{n,j}(x)$

$$H_{n,j}(x) = [1 - 2(x - x_j) L'_{n,j}(x_j)] L^2_{n,j}(x), \quad \hat{H}_{n,j}(x) = (x - x_j) L^2_{n,j}(x).$$

To prove it, evaluate at xi and recall that $L_{n,j}(x_i)=\delta_{ij}$.

Evaluation of functions: series

Many times the only way to evaluate a function is through the summation of a power series. Unless the series is more or less rapidly convergent. Even in this case it is useful to have at hand a method to accelerate the convergence of the series.

Aitken's Δ^2 acceleration,

Given a succession p_n that is at least linearly convergent to a value p, consider an n sufficiently large that,

$$\frac{p_{n+1}-p}{p_n-p} \approx \frac{p_{n+2}-p}{p_{n+1}-p}$$

We get, $$p_{n+1}^{\,2} + p^2 - 2p_{n+1}p \approx p_n p_{n+2} - p(p_{n+2}+p_n) + p^2$$

Or, $$p \approx \frac{p_n p_{n+2} - p_{n+1}^{\,2}}{p_{n+2} - 2p_{n+1} + p_n} \approx p_n - \frac{(p_{n+1}-p_n)^2}{p_{n+2}-2p_{n+1}+p_n}$$

Aitken's method consists in considering the succession p'_n, which converges faster than p_n,

$$p'_n = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}$$

```
In[1]:=
p[n_] := Cos[1/n]
pp[n_] := p[n] - (p[n+1] - p[n])^2/(p[n+2] - 2p[n+1] + p[n])

In[3]:= Do[Print[N[p[n]], "    ", N[pp[n]]], {n, 1, 7}]
0.540302    0.961775
0.877583    0.982129
0.944957    0.989786
0.968912    0.993416
0.980067    0.99541
0.986143    0.99662
0.989813    0.997408
```

Notation: $\Delta p_n = p_{n+1} - p_n$ Aitken: $p'_n = p_n - \frac{(\Delta p_n)^2}{\Delta^2 p_n}$

Some pesky trivial things:

Polynomials:

One should never write: `p=c(1)+c(2)*x+c(3)*x**2+c(4)*x**3`

One should write: `p=c(1)+x*(c(2)+x*(c(3)+x*c(4)))`

Which can be economically coded as,

```
p=c(n)
dp=0.
do ll  j=n-1,1,-1
     dp=dp*x+p
     p=p*x+c(j)
enddo ll
```

Which yields the derivative of the polynomial as well.

Complex arithmetic:

Most modern compilers allow to handle complex numbers as straightforwardly as real numbers. Complex arithmetic however, can lead to unexpected pitfalls, normally overflows and underflows in intermediate operations.

Example: $|a+ib|=\sqrt{a^2+b^2}$

If a or b are large, yet well within machine precision, it could happen that a^2 or b^2 are not (e.g. $a=10^{20}$).

The correct way to code this is,

$$|a+ib|=\begin{cases}|a|\sqrt{1+(b/a)^2} & |a|>|b| \\ |b|\sqrt{1+(a/b)^2} & |a|<|b|\end{cases}$$

Similar tricks are needed for division of complex numbers, square roots, etc. (See Numerical Recipes)

Recurrence relations:

Many useful functions satisfy recurrence relations,

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$$

$$J_{n+1}(x) = \frac{2n}{x}J_n(x) - J_{n-1}(x)$$

$$nE_{n+1}(x) = e^{-x} - xE_n(x)$$

And one might want to use them to generate values of the functions. As we mentioned in the first class, one has to be careful about roundoff error in these constructions. If the recurrence relation has two solutions f_n and g_n such that f_n/g_n->0 when n is large, then one is out of luck. The first steps get "contaminated" with the growing solution due to roundoff error and that solution grows out of control (stiffness).

The second relation listed above suffers this problem.

To study relations of this sort, one starts by considering relations similar to the ones of interest but with constant coefficients, for instance for the one for the Bessel functions,

$$y_{n+1} - 2\gamma\, y_n + y_{n-1} = 0$$

Where $\gamma = n/x$ One then tries a solution $y_n = a^n$ which gives,

$$a^2 - 2\gamma\, a + 1 = 0 \qquad a = \gamma \pm \sqrt{\gamma^2 - 1}$$

The recurrence is stable if $|a|<1$. This holds, for the example of interest only if $n<x$.

Clenshaw's formula:

Provides an elegant way of summing coefficients times functions that satisfy a certain recurrence relation.

$$f(x) = \sum_{k=0}^{N} c_k F_k(x) \qquad F_{n+1}(x) = \alpha(n,x)F_n(x) + \beta(n,x)F_{n-1}(x)$$

Define now the quantities, $(k = N, N-1, \ldots, 1)$

$$y_{N+2} = y_{N+1} = 0$$

$$y_k = \alpha(k,x)y_{k+1} + \beta(k+1,x)y_{k+2} + c_k$$

If one now solves the last equation for c_k, and then substitutes in the original summation, one gets something like,

$$
\begin{aligned}
f(x) = & \cdots \\
& + [y_8 - \alpha(8,x)y_9 - \beta(9,x)y_{10}]F_8(x) \\
& + [y_7 - \alpha(7,x)y_8 - \beta(8,x)y_9]F_7(x) \\
& + [y_6 - \alpha(6,x)y_7 - \beta(7,x)y_8]F_6(x) \\
& + [y_5 - \alpha(5,x)y_6 - \beta(6,x)y_7]F_5(x) \\
& + \cdots \\
& + [y_2 - \alpha(2,x)y_3 - \beta(3,x)y_4]F_2(x) \\
& + [y_1 - \alpha(1,x)y_2 - \beta(2,x)y_3]F_1(x) \\
& + [c_0 + \beta(1,x)y_2 - \beta(1,x)y_2]F_0(x)
\end{aligned}
$$

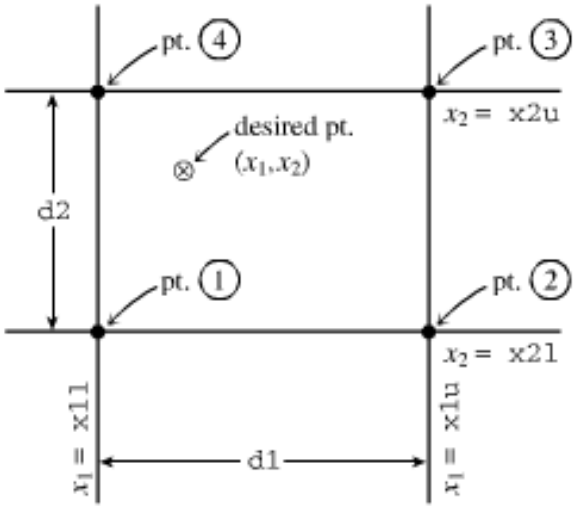
$$F_{n+1}(x) = \alpha(n,x)F_n(x) + \beta(n,x)F_{n-1}(x)$$

The only terms left are, $f(x) = \beta(1,x)F_0(x)y_2 + F_1(x)y_1 + F_0(x)c_0$

So one uses the formula defining the y's to "sweep down" to y_2 and y_1 and one is done.

Interpolations in higher dimensions:

No new conceptual ideas, just generalize previous concepts to each dimension.



$$t \equiv (x_1 - \texttt{x1a(j)})/(\texttt{x1a(j+1)} - \texttt{x1a(j)})$$

$$u \equiv (x_2 - \texttt{x2a(k)})/(\texttt{x2a(k+1)} - \texttt{x2a(k)})$$

$$y(x_1, x_2) = (1-t)(1-u)y_1 + t(1-u)y_2 + tuy_3 + (1-t)uy_4$$

Summary

- Splines are good interpolants for physics as they have up to second derivatives continuous.
- Series convergence can be accelerated.
- Tricks to deal with complex numers and polynomials.
- Recurrence relations: check for stiffness.