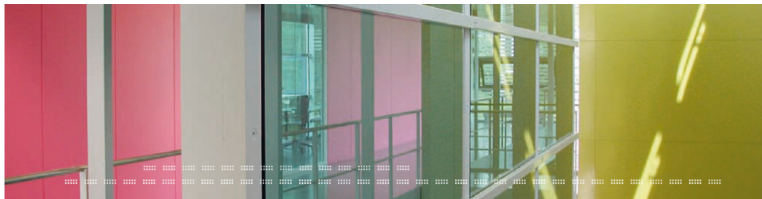
Linear Equation Systems and Interpolation

Dipl.-Inf. Domenic Jenz (jenz@hlrs.de)
HLRS, University of Stuttgart

27. January 2010



Outline

Direct solution of linear equation systems
- Pivoting
- Cholesky Decomposition

Interpolation
- Preparatory Remarks
- Interpolation with Polynomials
- Aitken-Neville
- Divided Differences
- Interpolation Error

Pivoting

- In the last codes we quite carelessly divided by diagonal elements, e.g. $r_{j,j}$. What if $r_{j,j} = 0$?
- *partial pivoting* or *column pivoting*:
  if $r_{j,j} = 0$ look for an entry $a_{i,j} \neq 0, i = j+1, ..n$
  - there is no such $a_{i,j}$ $\rightarrow$ completely vanishing column $\rightarrow$ det(A) = 0.
  - there are such $a_{i,j}$ $\rightarrow$ choose the one with maximum absolute value (let it be $a_{k,j}$) and exchange the rows k and j (also right-hand side)
  - large pivot $\rightarrow$ small elimination factors $l_{i,j}$.
- *total pivoting*:
  search not only in column j but in the whole remaining submatrix. As columns may also have to be exchanged and thus **x** may have to be reordered, it is more expensive.

Cholesky Decomposition I

For special matrices there are sometimes better algorithms.
One special type are the positive definite matrices.

- A quadratic symmetric matrix $A \in \mathbb{R}^{n,n}$ is called positive definite, if all of its eigenvalues are positive or if

$$x^T A x > 0 \quad \forall x \neq 0$$

- In $A = LR$, the factor R can be further decomposed into a diagonal matrix D and a upper triangular $\tilde{R}$ with ones on the diagonal:

$$A = L \cdot R = L \cdot D \cdot \tilde{R} \quad , D = diag(r_{1,1}, \cdots, r_{n,n})$$

The symmetry of A gives us:

$$A^T = (L \cdot D \cdot \tilde{R})^T = \tilde{R}^T \cdot D \cdot L^T = L \cdot D \cdot \tilde{R} = A$$

Cholesky Decomposition II

- Because the decomposition is unique:

$$L = \tilde{R}^T \quad \Rightarrow \quad A = L \cdot D \cdot L^T =: \tilde{L} \cdot \tilde{L}^T$$

- In the last step D is equally distributed among L and L^T ($D = \tilde{D} \cdot \tilde{D}$ with $\tilde{D} = diag(\sqrt{r_{1,1}}, \cdots, \sqrt{r_{n,n}})$).
- We rename $L = \tilde{L}$ for simpler notation:

$$A = L \cdot L^T$$

- In an LR-decomposition now the computation of the $r_{i,k}, i \neq k$ can be avoided $\rightarrow$ half of memory requirements and computational costs.

Cholesky Decomposition Derivation I

As with the LR-decomposition let's begin with the matrix product (remember $l_{i,j} = 0$ for $i < j$ and $l^T_{i,j} = 0$ for $i > j$). We only need a lower triangular Matrix, thus $i \geq k$:

$$a_{i,k} = \sum_{j=1}^{n} l_{i,j} \cdot l^T_{j,k} = \sum_{j=1}^{k} l_{i,j} \cdot l^T_{j,k} = \sum_{j=1}^{k} l_{i,j} \cdot l_{k,j}, \quad i \geq k$$

For $i = k$ (the diagonal elements) this gives:

$$a_{k,k} = \sum_{j=1}^{k} l^2_{k,j} \quad \rightarrow \quad l_{k,k} = \sqrt{a_{k,k} - \sum_{j=1}^{k-1} l^2_{k,j}}$$

Cholesky Decomposition Derivation II

For $i > k$:

$$a_{i,k} = \sum_{j=1}^{k} l_{i,j} \cdot l_{k,j} \quad \rightarrow \quad l_{i,k} = \left(a_{i,k} - \sum_{j=1}^{k-1} l_{i,j} \cdot l_{k,j} \right) / l_{k,k}$$

You need the elements in the columns with index < k for the current row (i) and a previous row (k).
Thus when we compute L column by column (starting at diagonal), we are fine, as every needed value for $l_{i,k}$ has been computed.

Cholesky Decomposition Code

```
1  def cholesky(A,n):
2    L = []
3    for i in range(0,n):
4      L.append([0]*n)
5    for k in range(0,n):
6      L[k][k] = A[k][k]
7      for j in range (0,k):
8        L[k][k] = L[k][k] - pow(L[k][j],2)
9      L[k][k] = pow(L[k][k], 0.5)
10     for i in range (k+1,n):
11       L[i][k] = A[i][k]
12       for j in range(0,k):
13         L[i][k] = L[i][k] - L[i][j]*L[k][j]
14       L[i][k] = L[i][k] / L[k][k]
15   return L
```

Approximation and Interpolation

- **approximation**: approximate some function $f(x)$ by some $p(x)$ which is easy to handle, following certain rules.
- **interpolation**: special case of the approximation, where the approximant has to go through a set of given points.
  - **interpolation of a function**: The given points are taken from the function. In the following we assume, that the functions are sufficiently smooth (all necessary derivatives shall exist).
  - **interpolation of discrete data**: You are given only some data points (e.g. from a measurement).

Basics and some notation

- **nodes**: the abscissas x_i, where the interpolant is to be "fixed"
- **nodal points**: pairs (x_i, y_i), i.e. nodes x_i with corresponding nodal values y_i
- polynomials are widespread interpolants:
  - $\mathbb{P}_n$: vector space of all polynomials with real coefficients of degree $\leq$ n.
  - dim($\mathbb{P}_n$) = n+1
  - with the differential operator D^k (for $\frac{\partial^k}{\partial x^k}$): $D^{n+1}p = 0 \;\; \forall p \in \mathbb{P}_n$
- different variants:
  - **single nodes:** for x_i prescribe $p(x_i)$ (*Lagrange Interpolation*)
  - **multiple nodes:** also derivatives are prescribed. (*Hermite Interpolation*)

Interpolation with Polynomials

- $p \in \mathbb{P}_n$ is called *polynomial interpolant* of f with respect to the nodes $a = x_0 < x_1 < \cdots < x_n = b$, if $p(x_i) = f(x_i) =: y_i \quad \forall i \in [0, n]$.
- The number of nodes determines the degree of the polynom.
- Existence and Uniqueness of the solution of this interpolation problem is known from analysis.
- The difference $f(x) - p(x)$ is called *remainder* or *error term*.

Approaches

$$p(x) := \sum_{i=0}^{n} a_i \cdot x^i$$

Make the incidential test for each node and solve the resulting system of $n+1$ linear equations.

- With the Lagrange polynomials

$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i} \quad \Rightarrow \; L_k(x_j) = \delta_{k,j}$$

you get:

$$p(x) = \sum_{k=0}^{n} y_k \cdot L_k(x)$$

Aitken-Neville's Scheme

Often, we don't need the interpolant p, but the evaluation p(x) at one or several points.

- define auxiliary polynoms $p_{i,k}(x)$ of degree k, which interpolate the nodes (x_l, y_l) for $l = i, ..., i + k$
- do this recursively with

$$p_{i,k}(x) := \frac{x_{i+k} - x}{x_{i+k} - x_i} \cdot p_{i,k-1}(x) + \frac{x - x_i}{x_{i+k} - x_i} \cdot p_{i+1,k-1}(x)$$

- Starting values are $p_{0,0}(x) = y_0, \ldots, p_{n-1,0}(x) = y_{n-1}$
- The value $p_{0,n-1}(x)$ will be the result you wanted.

Aitken-Neville Code

```
1  def aitken (x,y,x0,n):
2    x0 = float(x0)
3    p = []
4    for i in range(0,n):
5      p.append([0]*n)
6
7    for i in range(0,n):
8      p[i][0] = y[i]
9    for k in range (1,n):
10     for i in range(0, n-k):
11       p[i][k] = (x[i+k]-x0) / (x[i+k]-x[i]) * p[i][k-1]
12                 (x0-x[i])/(x[i+k]-x[i])*p[i+1][k-1]
13   return p[0][n-1]
```

Divided Differences I

Let us denote the highest coefficient of $p_{i,k}$ as:

$$[x_i, \ldots, x_{i+k}]f$$

and call it *divided differences of f of order k with respect to* $x_i, \ldots, x_{i+k}$, i.e.

$$p_{i,k}(x) = [x_i, \ldots, x_{i+k}]f \cdot x^k + r \qquad , r \in \mathbb{P}_{k-1}.$$

So we have:

$$[x_i]f = f(x_i) = y_i$$

$$[x_i, x_{i+1}]f = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Generally (with Aitken-Neville):

$$[x_i, \ldots, x_{i+k}]f := \frac{[x_{i+1}, \ldots, x_{i+k}]f - [x_i, \ldots, x_{i+k-1}]f}{x_{i+k} - x_i}$$

Divided Differences in Python

```
1  def divdiff (xList, yList, xVal):
2    length = len (xList)
3    diffs = []
4    for i in range (0, length):
5      diffs.append (yList[i])
6    for i in range (1,length):
7      for j in range (length-i, length):
8        lowerIndex = length - i - 1
9        diffs[j] = (diffs[j] - diffs[j-1])/(xList[j]-xList
10   retVal = diffs[0]
11   product = 1
12   for i in range (1, length):
13     product *= xVal - xList[i-1]
14     retVal += diffs[i] * product
15   return retVal
```

Newton's interpolation formula

The used basis for $\mathbb{P}_n$ is
$\{1, x - x_0, (x - x_0) \cdot (x - x_1), (x - x_0) \cdot \ldots \cdot (x - x_{n-1})\}$.
The approach here will be:

$$p_{0,k}(x) = \sum_{i=0}^{k} c_i \left(\cdot \prod_{j=0}^{i-1} (x - x_j) \right)$$

As c_k is the highest coefficient of $p_{0,k}$:

$$c_k = [x_0, \ldots, x_k]f$$

Adding one node will only add another summand:

$$p_{0,k+1} = \sum_{i=0}^{k+1} c_i \left(\cdot \prod_{j=0}^{i-1} (x - x_j) \right) = p_{0,k}(x) + c_{k+1} \cdot \prod_{j=0}^{i-1} (x - x_j)$$

Newton's interpolation formula II

As adding a new node only adds a new summand, the previously calculated c_k don't change, and in the end we get *Newton's interpolation formula*:

$$
\begin{aligned}
p_{0,n}(x) = p(x) \quad = \quad & [x_0]f + \\
& [x_0, x_1]f \cdot (x - x_0) + \\
& [x_0, x_1, x_2]f \cdot (x - x_0) \cdot (x - x_1) + \\
& [x_0, \ldots, x_3]f \cdot (x - x_0) \cdot (x - x_1) \cdot (x - x_2) + \\
& \vdots \\
& [x_0, \ldots, x_n]f \cdot \prod_{i=0}^{n-1} (x - x_i)
\end{aligned}
$$

Interpolation Error I

Now we estimate the interpolation error $|f(x) - p(x)|$.

- Let $f \in \mathcal{C}^n([x_0, x_n])$,i.e. n times continuously differentiable on $[x_0, x_n]$. Then there exists a $\xi \in [x_0, x_n]$ with:

$$[x_0, \ldots, x_n]f = \frac{D^n f(\xi)}{n!}$$

- Now consider some $\tilde{x} \in [x_0, x_n] \setminus \{x_0, \ldots, x_n\}$ and let $\tilde{p}(x) \in \mathbb{P}_{n+1}$ interpolate also $\tilde{x}$ in addition to $x_0, \ldots, x_n$:

$$f(\tilde{x}) - p(\tilde{x}) = P(\tilde{x}) - p(\tilde{x}) = [x_0, \ldots, x_n, \tilde{x}]f \cdot \prod_{i=0}^{n} (\tilde{x} - x_i)$$

$$= \frac{D^{n+1} f(\xi)}{(n+1)!} \cdot \prod_{i=0}^{n} (\tilde{x} - x_i)$$

Interpolation Error II

For equidistant nodes with mesh width $h := x_{i+1} - x_i$, the product can be estimated.
In the worst case the distance of $\tilde{x}$ to the farest node is nh; to the second farest $(n-1)h$ and so on.
Thus we can estimate:

$$|f(\tilde{x}) - p(\tilde{x})| \leq \frac{\max_{[x_0,x_n]} |D^{n+1} f(x)|}{n+1} \cdot h^{n+1} \in \mathcal{O}(h^{n+1})$$