Development with the Matrix package

Eddy Offermann
eddy@rentec.com

Which matrix/vector class

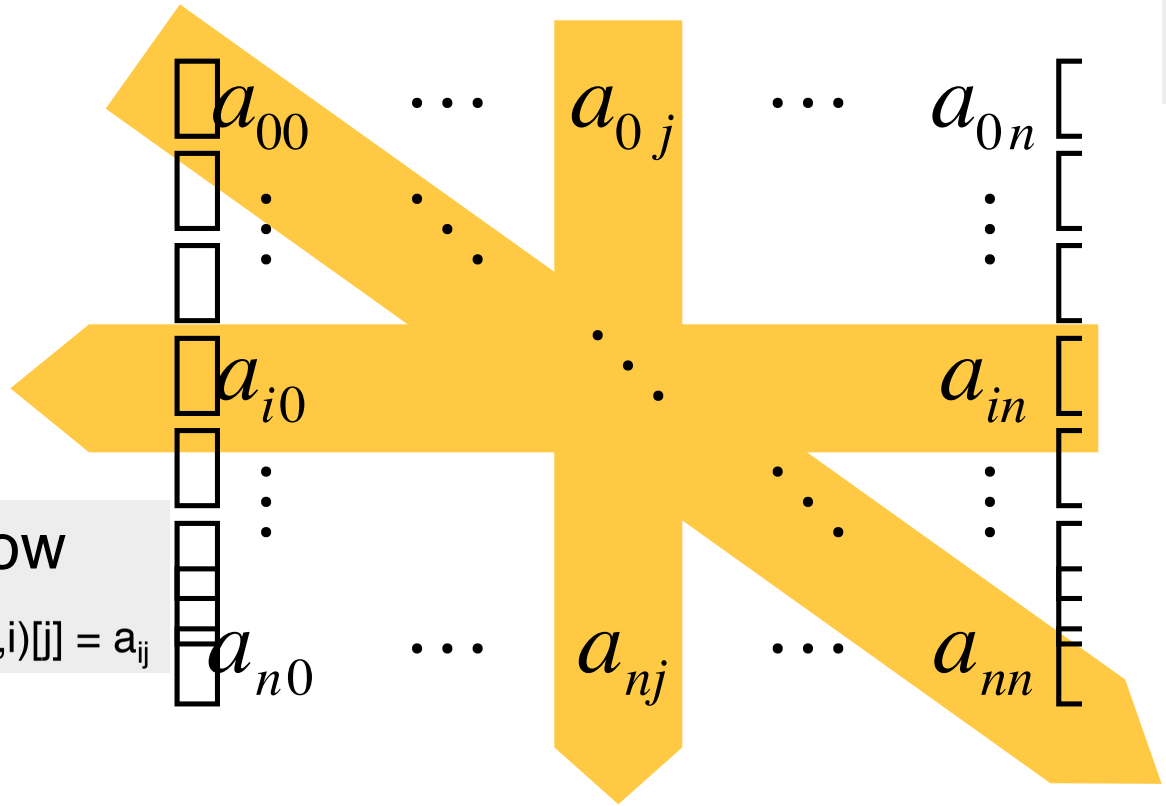
May 2001 :”Jacek M. Holeczek": "The question of a linear algebra package in ROOT was already raised a couple of times without any conclusion.”

- what ?
  Reasonably complete package integrated in Root/Cint
- what not ?
  Rewrite of LAPACK++
- so what ?
  Extend existing TMatrix/TVector (Oleg E. Kiselyov (oleg@pobox.com)

Matrix Access

A[i][j] = a_{ij}

A(i,j) = a_{ijj}



TMatrixRow

TMatrixRow(A,i)[j] = a_{ij}

TMatrixColumn

TMatrixColumn(A,j)[i] = a_{ij}

TMatrixDiag

TMatrixDiag(A)[i] = a_{ii}

Data transfer

- Usual suspects between/among TMatrix and TVector
- Direct data access

Careful, column-wise stored

```
void TMatrix::SetElements(const Float_t *elements,Option_t *option)

void GetElements(Float_t* elements, Option_t* option) const

Float_t* GetElements()
```

- TLazyMatrix

```
class TMyMatrix : public TLazyMatrix {
.......
private:
  void FillIn(TMatrix &m) const;
};

inline TMatrix::TMatrix(const TLazyMatrix &lazy_constructor)
{
  .......
  lazy_constructor.FillIn(*this);
}
```

Nice example in
$(ROOTSYS)/test/vlazy.cxx

Functionality

- Most binary operations among/between matrices and vectors
- Matrix inversion
- Eigen-values and -vectors

Extendable

- Write operations without changing class

```
class NormMatrixD : public
    TElementPosActionD
{
  const Double_t *fA;
  void Operation(Double_t &element)
  { Double_t val =
    TMath::Sqrt(TMath::Abs(fA[fI]*fA[fJ]));
    if (val != 0.0) element /= val;
  public:
   NormMatrixD(const Double_t *a)
     { fA = a; }
};
```

- Apply operation (normalizing before inversion)

```
TArrayD diag;
……
NormMatrixD norm(diag.GetArray());

x.Apply(norm);
TMatrixD xInv(TMatrixD::kInverted,x);
xInv.Apply(norm);
```

Example: solve normal equation

```
TVectorD NormalEq(TMatrixD &a,TVectorD &c,
                  TVectorD &variance)
{
  // Find optimal solution for (A x - c)^T (A x - c) = 0

  // hessian (A^T A)
  Double_t det;
  TMatrixD ataInv(a,TMatrixD::kTransposeMult,a);
  ataInv.Invert(&det);

  // gradient (A^T c)
  TMatrixD at(TMatrixD::kTransposed,a)
  TVectorD solution = c;
  solution *= at;

  // solution = (A^T A)^-1 A^T c
  solution *= ataInv;
  variance.ResizeTo(solution.GetNoElements());
  variance = TMatrixDDiag(ataInv);

  return solution;
}
```

Fancy constructors minimize moving data around

Pseudo Inverse

$$A^{+} = \left(A^{T}A\right)^{-1}A^{T}$$

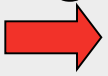
What is missing ?

- Template classes

```
<template class T>  class TMatrixTempl
<template class T>  class TVectorTempl
Typedef TMatrixTempl<Float_t> TMatrix
Typedef TMatrixTempl<Double_t> TMatrixD
```

- New functionality
  - Single Value Decomposition
  - Pseudo Inverse
  - ??

Applying Root at a Hedge Fund

- Renaissance Technologies :
  - 180 employees, ~50 PhD's (math, physics, comp. science….)
  - Technical trading:
    data into computer  trade recommendation

- Same data analysis issues as HEP:
  - Ten's of Gbytes of data / day
  - Data storage / reduction

Comparison

Finance Experiment

- 24hrs / 5days
- Several data suppliers: different snapshots
- Reduce data in analysis: use only trade prices, not bid/asks
- Test new ideas for price prediction in a simulator
- Events are serially correlated, consequences for parallelization of simulation/analysis

Particle Accelerator Exp

- 24hrs / 7days
- Several detector stacks: try to get consistent particle tracks
- Reduce data by not storing all drift times but particle's id and impulse
- Design experiment through simulations in GEANT
- Scattering events are independent; each event shipped to different cpu box

Simulator

```
TTree rawData;
TS *sources = new TS(stimey,etime);
Double_t PandL = 0;
for (Int_t time = stime; time < etime;
     time+=step)
{
  Prepare_data(rawData,sources);

  TArrayD modelPar =
     Fit_Model_Par(sources,time);

  TArrayD predPrice =
     Make_Prediction(sources,modelPar,time);

  TArrayD betPos  =
     Make_Bet(predPrice,sources,time);


  PandL += Accounting(betPos,sources,time);
}
Cout << ((PandL>0) ?
  “buy car” : “sell car”) <<endl;
```

$$\sum_j (\text{actualPrice}_j - \text{predict}(\text{modelPar},\text{sources}))^2$$

Be careful no cheats !

$$\sum_i \text{predPrice}_i \cdot \text{betPos}_i - Risk(\text{betPos}_i)$$

$$\sum_i \text{actualPrice}_i \cdot \text{betPos}_i$$

Root/Cint in High-Frequency Finance

- storing / filtering data
- tracking patterns through visualization and statistical tools
- Root/Cint macros for quick proto-typing of new predictive sources `gROOT->ProcessLine(command,&error)`
- PROOF to parallelize simulations

– An Introduction to High-Frequency Finance,
Michel Dacorogna et al., Academic Press

– www.olsen.ch/research/index.html