



Amsterdam Optimization Modeling Group LLC

Erwin Kalvelagen

*erwin@amsterdamoptimization.com*

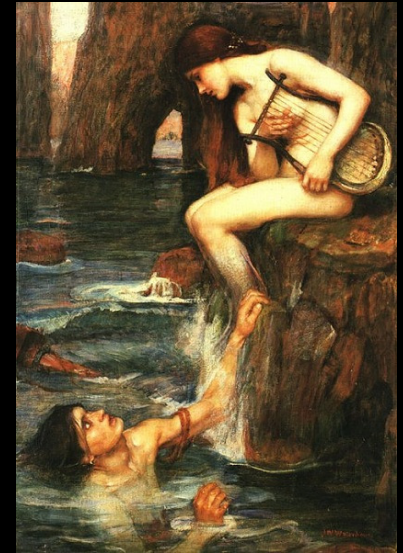
**DATA AND SOFTWARE  
INTEROPERABILITY WITH GAMS:  
A USER PERSPECTIVE**

# Modeling Languages

- Specialized Modeling Languages (GAMS, AMPL,...) are very good in what they do
  - Efficient, compact representation of a model
  - In a way that allows thinking about the complete model
  - And that allows and encourages experiments and reformulations
  - Through maintainable models
  - Especially when they become really big
  - Handle irregular and messy data well

# API attraction: Σειρήν

- Many new users are seduced to program against solver API's
- Familiar environment, no new funny language to learn
- But for large, complex models this is really a step back
- Even when using higher level modeling-fortified API's



# Modeling Environments

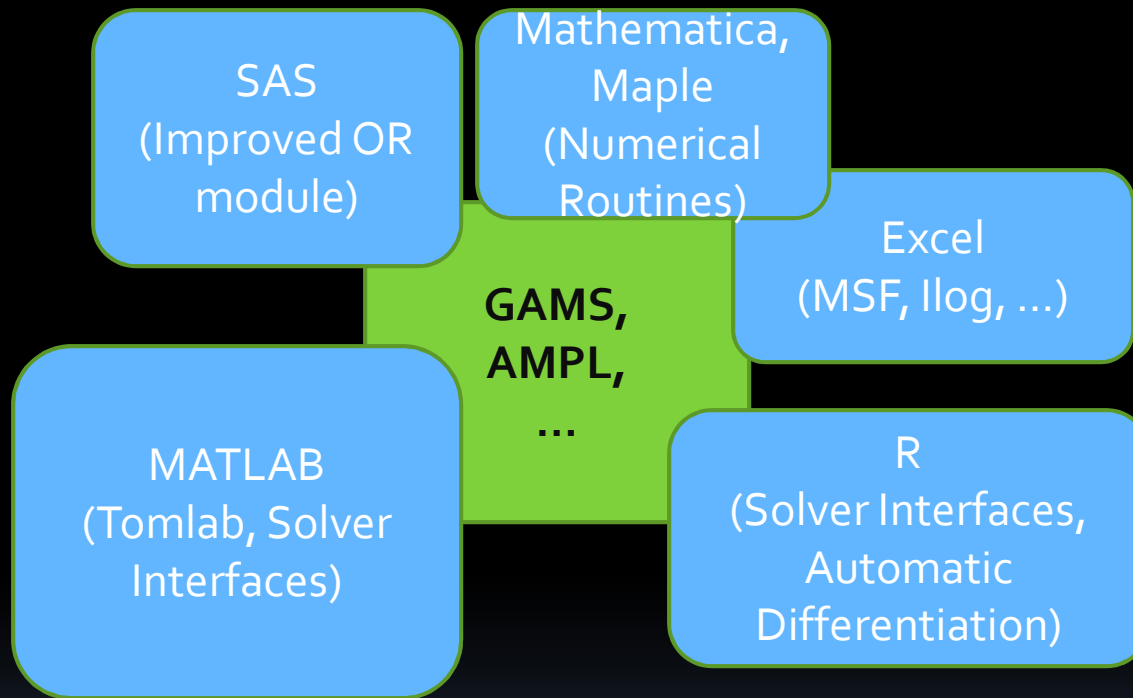
Environment	Solver API	Modeling API	Modeling Language
GAMS, AMPL			X
IBM ILOG	Cplex API	Concert	OPL
MS Solver Foundation	Solver Level API	SFS	OML
GLPK	API		Mathprog

Some programmers love:

Malloc, Linker errors, Compiler versions, Library versions,  
Makefiles, Windows vs. Unix, Debuggers, ...

But for others this is time taken away from modeling...

# Also increased use of optimization in ...



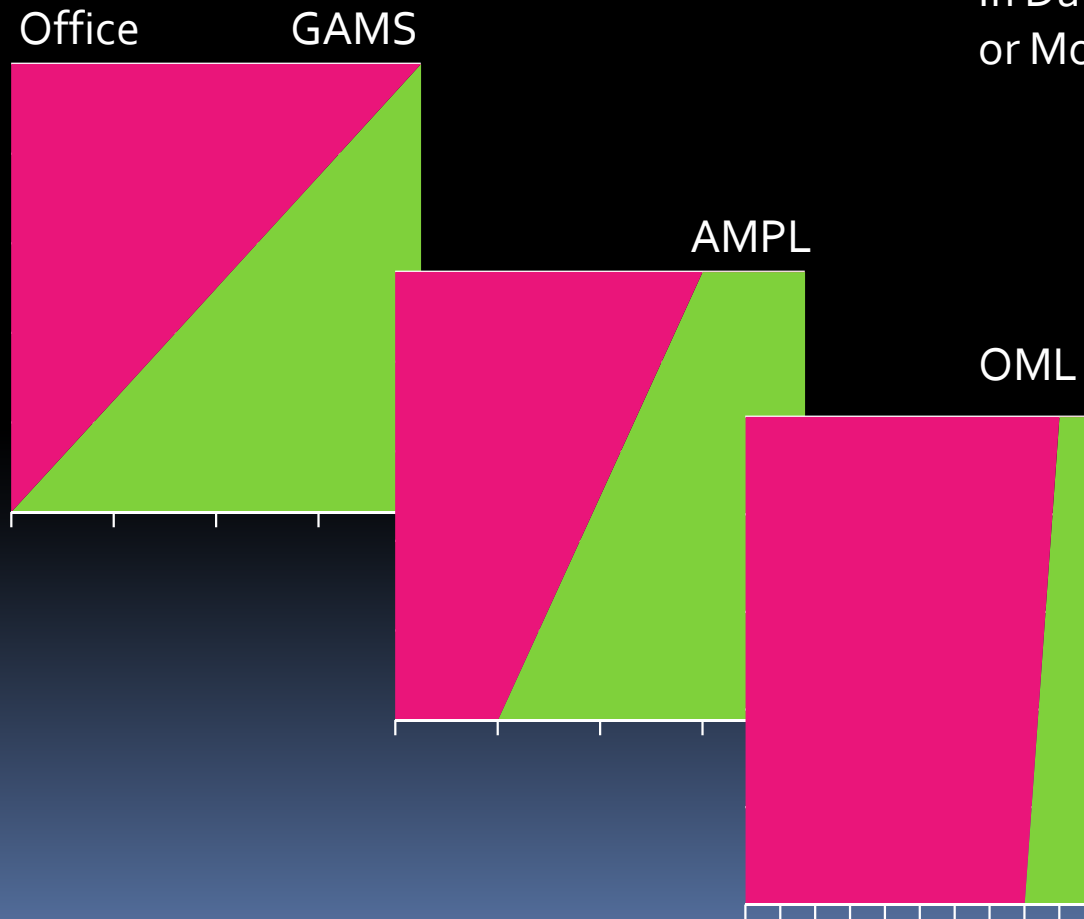
# Answer: Interoperability

- Make GAMS more attractive for programmers by allowing to use external software
- Make the user decide what to do in GAMS or in other environment
- Make data exchange as easy as possible
  - Even for large data
  - Safety: this is a spot where lots of things can and will go wrong

# Flexibility

- Do not decide for user
  - Data manipulation can be done in GAMS or Excel
  - Computation can be done in GAMS or external software
  - Allow these decisions to be made depending on the situation
    - Skills
    - Available software
    - Suitability

# E.g. Data handling



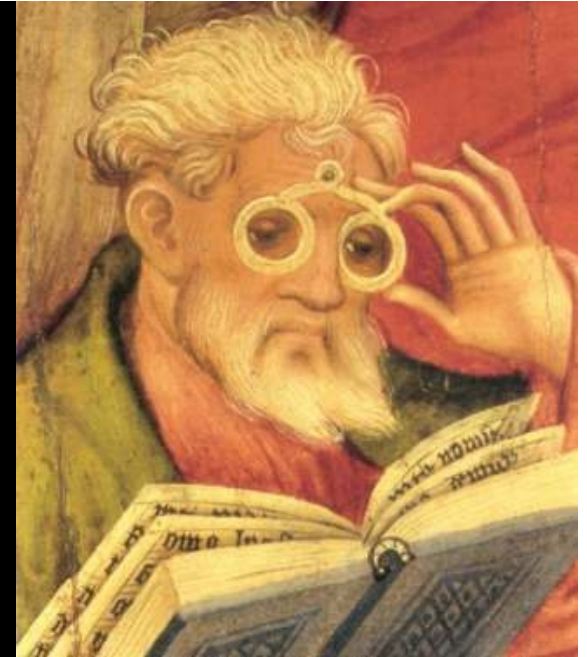
Where to put functionality:  
In Data Source Environment  
or Modeling System

This has also to do  
with procedural  
vs declarative and  
with data  
manipulation  
capabilities



# GDX in practice

- It really works
  - Binary, fast
  - You can look at it
- Limitations:
  - Cannot add records or symbols
    - Eg: combine two gdx files
    - GDX is immutable
  - Not self contained wrt GAMS:
    - Needs declarations
  - Zero vs non-existent
  - GAMS interface
    - \$load is dangerous
    - Compile time vs execution time
    - Execution time limits (each call separate gdx, cannot add set elements at execution time)



# Example: USDA Farm database

- Imports 3 MDB + 1 XLS file → 1 GDX file
  - ± 5 million records (raw data)

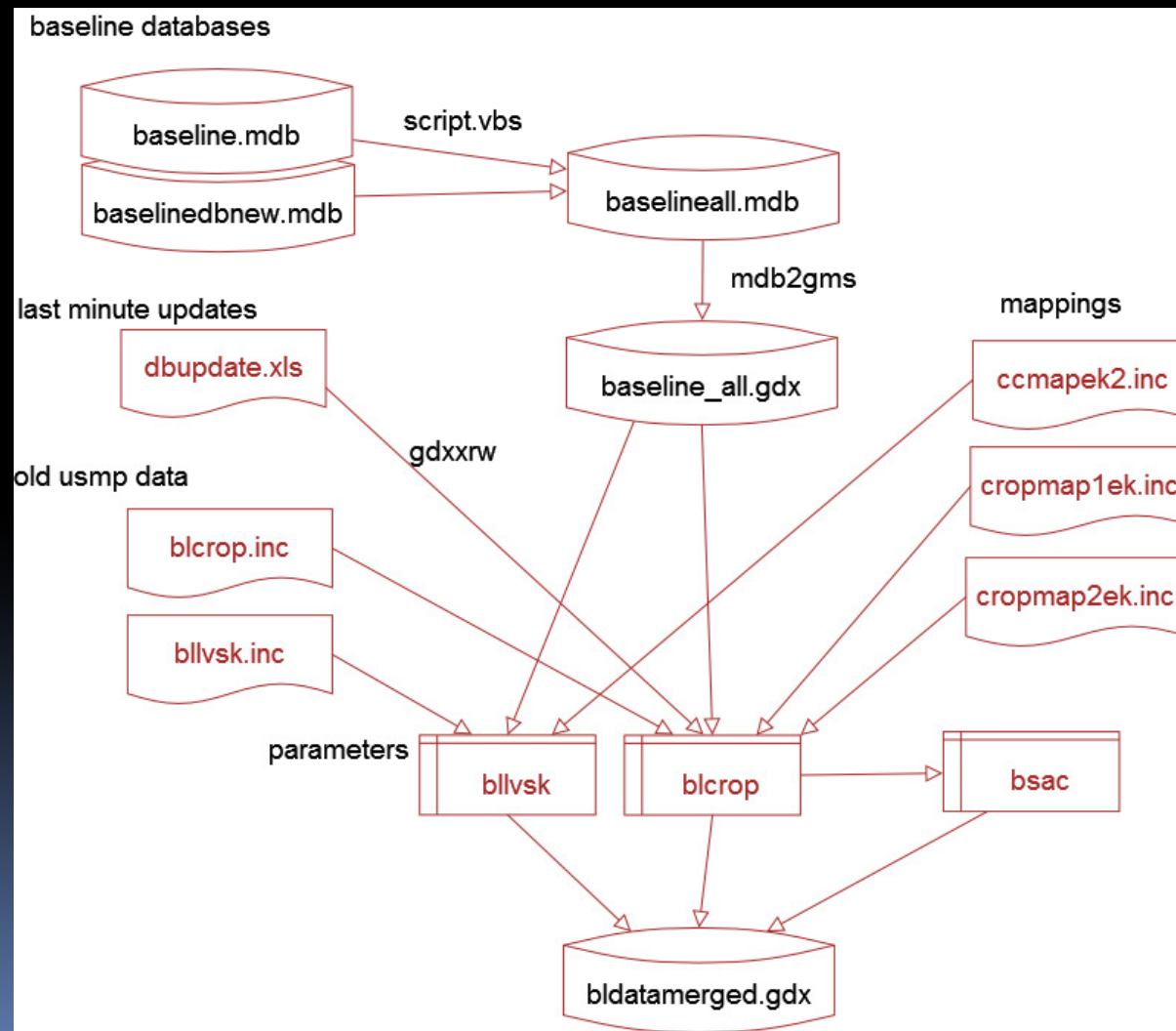
165 symbols

File	Size (bytes)
FARMLandWaterResourcesDraft.mdb	80,650,240
FARMResourcesProductionDraft.mdb	429,346,816
GTAP54-NonLandIntensive.mdb	303,616,000
FIPS2&FAOCountries.xls	29,184
farm.gdx	119,811,434
farm.gdx (compressed)	52,924,664

- Good, compact way to distribute and reuse large data sets (input or output)
- Aggregation easier in GAMS than in Access!

# Example: USDA Reap Model

- Combine data from different sources



# Conversion mdb -> gdx

```
$onecho > cmd.txt
I=FeedGrainsData.mdb
X=FeedGrainsData.gdx

q1=select commodity from tblCommodities
s1=commodity

q2=select attribute from tblAttributes
s2=attribute

q3=select period from tblTimePeriods
s3=period

q4=select unit from tblUnits
s4=unit

q5=select distinct(iif(isnull(isource),'blank',isource)) \
      from tblFG_update where \
      not isnull(year)
s5=isource

q6=select geo from tblgeography
s6=geocode

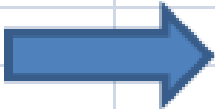
q7=select commodity,attribute,unit,iif(isnull(isource),'blank',isource),geocode,year,period,value \
      from tblFG_update where \
      not isnull(year)
p7=feedgrains
$offecho

$call mdb2gms @cmd.txt
```

# Typical Problems

- NULL's
- Duplicate records
- Multivalued tables
- More difficult processing:
  - Get latest available number
    - Difficult in SQL and in GAMS

Advantage of SQL: we can repair a number of problems on the fly.

comm	year	forecasy	value		comm	year	value
corn	2010	2006	12		corn	2010	12.5
corn	2010	2007	12.5		barley	2010	9.1
barley	2010	2005	9				
barley	2010	2006	9.1				

# Data manipulation

- Role of data manipulation in a modeling language
- OML
  - No data manipulation at all
  - Do it in your data source environment (e.g. Excel)
- AMPL
  - More extensive data manipulation facilities
  - Powerful if fits within declarative paradigm
- GAMS
  - Extensive use of data manipulation
  - Procedural

# Policy Evaluation Models

- Often have serious data handling requirements
  - Aggregation/disaggregation
  - Estimation/Calibration
  - Simulation
- Examples:

Model	LOC	LOC for equ's
Polysys	22576	< 500
Impact2000	17284	0
IntegratedIW5	20177	< 500

# Sparse Data: matrix multiplication

Ampl

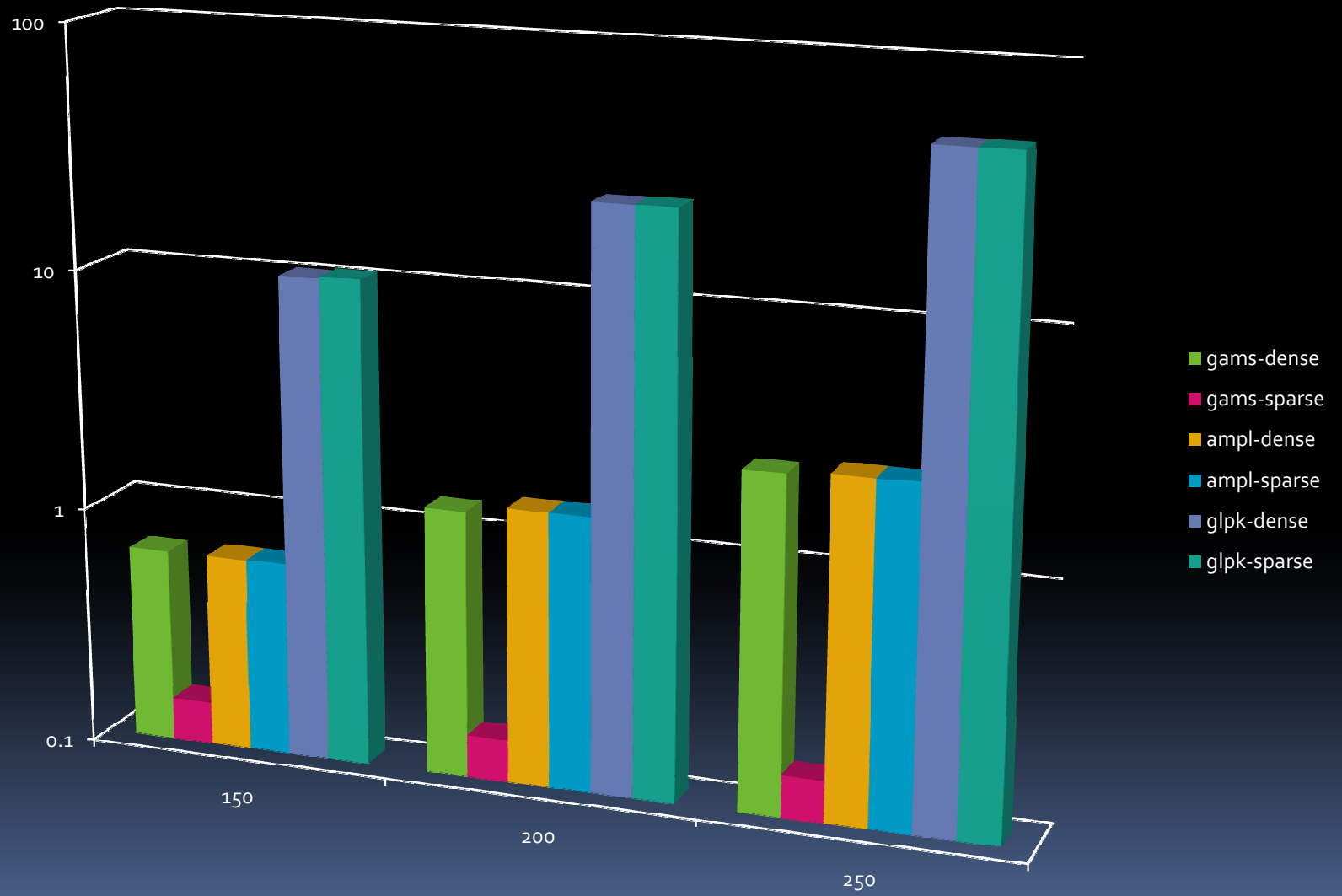
```
param N := 250;  
set I := {1..N};  
param A{i in I, j in I} := if (i=j) then 1;  
param B{i in I, j in I} := if (i=j) then 1;  
param C{i in I, j in I} := sum{k in I} A[i,k]*B[k,j];  
param s := sum{i in I, j in I} C[i,j];  
display s;  
end;
```

GAMS

```
set i /1*250/;  
alias (i,j,k);  
parameter A(i,j),B(i,j),C(i,j);  
A(i,i) = 1;  
B(i,i) = 1;  
C(i,j) = sum(k, A(i,k)*B(k,j));  
scalar s;  
s = sum((i,j),C(i,j));  
display s;
```



# Timings



# Solving Linear Equations

- Solve  $Ax=b$  for  $x$
- Often not a good idea to calculate  $A^{-1}$
- In GAMS we can solve by specifying  $Ax=b$  as equations

```
linsys(i).. sum(j, a(i,j)*x(j)) =e= b(i);
```

# Inverse

- If you really want the inverse of a matrix:

```
alias(i,j,k);  
  
parameter unity(i,j);  
unity(i,i)=1;  
  
variable inv(i,j);  
equation inverse(i,j);  
  
inverse(i,j).. sum(k, inv(i,k)*a(k,j)) =e= unity(i,j);
```

i.e. solve for  $A^{-1}$

$$A^{-1}A = I$$

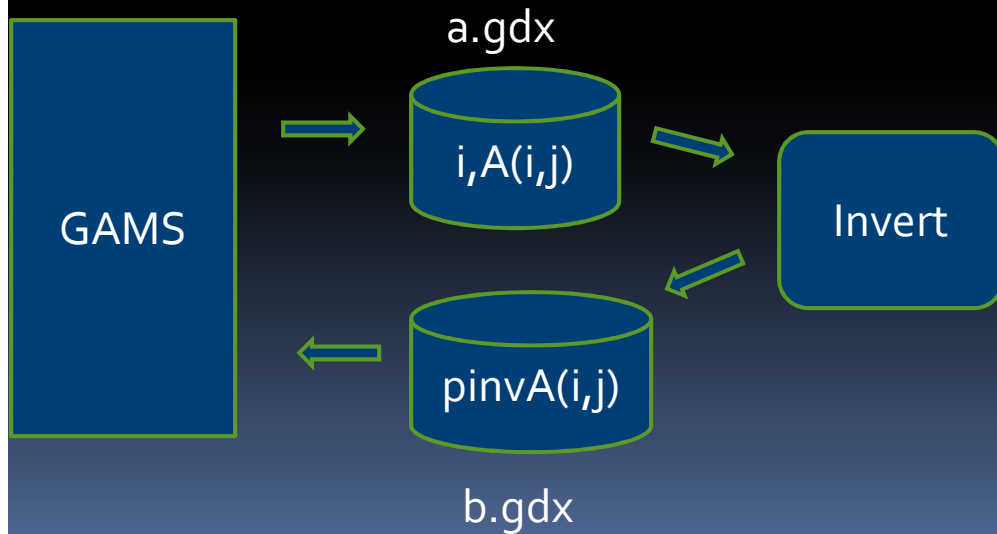
# Speed Up: Advanced Basis

- We can provide advanced basis so the calculation takes o Simplex iterations
  - $\text{Inv.m}(i,j) = 0$ ; (var: basic)
  - $\text{Inverse.m}(i,j) = 1$ ; (equ: non-basic)

	method=1	method=2
n=50	0.637	0.433
n=100	8.267	4.036
n=200	313.236	53.395

# External Solver using GDX

```
execute_unload 'a.gdx',i,a;  
execute '=invert.exe a.gdx i a b.gdx pinva';  
parameter pinva(i,j);  
execute_load 'b.gdx',pinva;
```




# Test Matrix: Pei Matrix

$$\begin{pmatrix} 1+\alpha & 1 & 1 & 1 \\ 1 & 1+\alpha & 1 & 1 \\ 1 & 1 & 1+\alpha & 1 \\ 1 & 1 & 1 & 1+\alpha \end{pmatrix}$$

	method=1	method=2	method=3
n=50	0.637	0.433	0.027
n=100	8.267	4.036	0.055
n=200	313.236	53.395	0.118



# Other tools

- Cholesky
  - Eigenvalue
  - Eigenvector
- 

# Max Likelihood Estimation

- NLP solver can find optimal values: estimates
- But to get covariances we need:
  - Hessian
  - Invert this Hessian
- We can do this now in GAMS
  - Klunky, but at least we can now do this
  - GDX used several times



# MLE Estimation Example

```
* Data:
* Number of days until the appearance of a carcinoma in 19
* rats painted with carcinogen DMBA.

set i /i1*i19/;
table data(i,*)
      days  censored
i1      143
i2      164
i3      188
i4      188
i5      190
i6      192
i7      206
i8      209
i9      213
i10     216
i11     220
i12     227
i13     230
i14     234
i15     246
i16     265
i17     304
i18     216      1
i19     244      1

      set k(i) 'not censored';
      k(i)$(data(i,'censored')=0) = yes;

      parameter x(i);
      x(i) = data(i,'days');

      scalars
        p 'number of observations'
        m 'number of uncensored observations'
      ;

      p = card(i);
      m = card(k);

      display p,m;
;
```

# MLE Estimation

```
*-----  
* estimation  
*-----  
  
scalar theta 'location parameter' /0/;  
  
variables  
    sigma 'scale parameter'  
    c     'shape parameter'  
    loglik 'log likelihood'  
;  
  
equation eloglike;  
  
c.lo = 0.001;  
sigma.lo = 0.001;  
  
eloglike.. loglik =e= m*log(c) - m*c*log(sigma)  
              + (c-1)*sum(k,log(x(k)-theta))  
              - sum(i,((x(i)-theta)/sigma)**c);  
  
model mle /eloglike/;  
solve mle maximizing loglik using nlp;
```

# Get Hessian

```
*-----  
* get hessian  
*-----  
option nlp=convert;  
$onecho > convert.opt  
hessian  
$offecho  
mle.optfile=1;  
solve mle minimizing loglik using nlp;  
  
*  
* gams cannot add elements at runtime so we declare the necessary elements here  
*  
set dummy /e1,x1,x2/;  
  
parameter h(*,*,*) '-hessian';  
execute_load "hessian.gdx",h;  
display h;  
  
set j /sigma,c/;  
parameter h0(j,j);  
h0('sigma','sigma') = h('e1','x1','x1');  
h0('c','c') = h('e1','x2','x2');  
h0('sigma','c') = h('e1','x1','x2');  
h0('c','sigma') = h('e1','x1','x2');  
display h0;
```

H: individual row Hessians

		x1	x2
e1	x1	0.0114575560266001	-0.0257527570747253
	x2		0.934221386133885

# Invert Hessian

```
*-----  
* invert hessian  
*-----  
  
execute_unload "h.gdx",j,h0;  
execute "=invert.exe h.gdx j h0 invh.gdx invh";  
parameter invh(j,j);  
execute_load "invh.gdx",invh;  
display invh;
```

# Normal Quantiles

```
* -----  
* quantile of normal distribution  
* -----  
  
* find  
*   p = 0.05  
*   q = probit(1-p/2)  
  
scalar prob /.05/  
  
* we don't have the inverse error function so we calculate it  
* using a small cns model  
equations e;  
variables probit;  
e.. Errorf(probit) =e= 1-prob/2;  
model inverterrorf /e/;  
solve inverterrorf using cns;  
  
display probit.l;  
  
* verification:  
*> qnorm(0.975);  
*[1] 1.959964  
*>
```

Or just use 1.96

# Finally: confidence intervals

```
*-----  
* calculate standard errors and confidence intervals  
*-----  
  
parameter result(j,*);  
result('c','estimate') = c.l;  
result('sigma','estimate') = sigma.l;  
  
result(j,'stderr') = sqrt(abs(invh(j,j)));  
  
result(j,'conf lo') = result(j,'estimate') - probit.l*result(j,'stderr');  
result(j,'conf up') = result(j,'estimate') + probit.l*result(j,'stderr');  
  
display result;
```

```
----      168 PARAMETER result  
  
           estimate      stderr      conf lo      conf up  
sigma      234.319        9.646        215.413        253.224  
c           6.083         1.068         3.989         8.177
```

# New developments

- Instead of calling external programs with a GDX interface
- Call a user provided DLL
- With simplified syntax:

```
Parameter A(i,j),B(j,i);
```

```
A(i,j) = ...
```

```
Scalar status;
```

```
BridgeCall('gamslapack', 'invert', A, B, Status);
```

# Behind the scenes

- Map GAMS data to fortran, c, ...
- Deal with calling conventions (stdcall)
- Specified in a small spec file

## file bridglibrary.ini

```
[bridge]
id=GAMS bridge library
lib1=gamslapack
lib2=gamsgsl
```

```
subroutine invert(a,n,b,info)
```

## beginning of file gamslapack.ini

```
[Library]
Version=1
Description=GAMS interface to LAPack
LibName=gamslapack
DLLName=gamslapack
Storage=F
```

## [invert]

```
name=invert
i1=Q // param 1 = square matrix
i1d=i2
i2=D // param 2 = n
o1=Q // param 3 = square matrix
o1d1=i1,2
o1d2=i1,1
o2=N // param 4 = info
status=o2
```



# Gets more exciting when...

- We can parse subroutine headers
- In libraries
- And generate this automatically
- This will open up access to
  - Numerical, statistical libraries
  - Current tools as function (sql2gms, LS solver,...)
  - Etc.
- Longer term: also for equations
  - derivatives