# Efficient Computation of the Characteristic Polynomial

J-G. Dumas, C. Pernet, Z. Wan
`{jgdumas,pernet}@imag.fr,wan@cis.udel.edu`

ISSAC, 27th July 2005

# Introduction

**Goal** Compute $det(\lambda I - A)$ over $\mathbb{Z}$

# Introduction

**Goal** Compute $det(\lambda I - A)$ over $\mathbb{Z}$

**Applications** Computational mathematics

- Matrix equivalence : via Frobenius normal form,
- Graph theory : cospectrality of graphs.

# Introduction

**Goal** Compute $det(\lambda I - A)$ over $\mathbb{Z}$

**Applications** Computational mathematics

- Matrix equivalence : via Frobenius normal form,
- Graph theory : cospectrality of graphs.

**Focus** on the design of algorithms

- Efficient in practice (time and memory)
- Mainly for dense matrices
- Probabilistic is enough if error probability $\epsilon \simeq 2^{-55}$

# Outline

# Sommaire

# Global considerations

- Take benefit of the huge efforts for numerical computations : architecture, B.L.A.S.

  ⇒cf. FFLAS & FFPACK

# Global considerations

- Take benefit of the huge efforts for numerical computations : architecture, B.L.A.S.
  $\Rightarrow$cf. FFLAS & FFPACK
- Design of block algorithms to rely on matrix multiplication

# Global considerations

- Take benefit of the huge efforts for numerical computations : architecture, B.L.A.S.
  $\Rightarrow$cf. FFLAS & FFPACK
- Design of block algorithms to rely on matrix multiplication

  Memory tuning better data locality, cache optimizations

# Global considerations

- Take benefit of the huge efforts for numerical computations : architecture, B.L.A.S.

  ⇒cf. FFLAS & FFPACK
- Design of block algorithms to rely on matrix multiplication

  Memory tuning better data locality, cache optimizations

  Fast algorithms into practice $\mathcal{O}(n^\omega)$ in theory but also proven useful in practice

# Sommaire

# Krylov's method

## Definition

$$K = [v|Av| \qquad \ldots \qquad |A^n v]$$

# Krylov's method

## Definition

$$K = [v|Av|\dots|A^d v|A^{d+1} v|\dots|A^n v]$$

# Krylov's method

## Definition

$$K = [\underbrace{v|Av|\dots|A^d v|}_{X} A^{d+1} v|\dots|A^n v]$$

# Krylov's method

## Definition

$$K = [\underbrace{v|Av|\dots|A^d v|}_{X} A^{d+1} v|\dots|A^n v]$$

$$\Rightarrow P_{\mathtt{min}}^{A,v}(X) = X^d - \sum_{i=1}^{d} y_i X^{i-1}$$

# Krylov's method

## Definition

$$K = [\underbrace{v|Av|\dots|A^d v|}_{X} A^{d+1}v|\dots|A^n v]$$

$$\Rightarrow P_{\mathtt{min}}^{A,v}(X) = X^d - \sum_{i=1}^{d} y_i X^{i-1}$$

## Fact

$$AX = XC_{P_{min}^{A,v}}$$

$$C_{P_{min}^{A,v}} = \begin{pmatrix} 0 & & & m_0 \\ 1 & 0 & & m_1 \\ & \ddots & \ddots & \vdots \\ & & 1 & m_{d-1} \end{pmatrix}$$

# Krylov's method

## Definition

$$K = [\underbrace{v|Av|\ldots|A^d v|}_{X} A^{d+1}v|\ldots|A^n v]$$

$$\Rightarrow P_{\mathtt{min}}^{A,v}(X) = X^d - \sum_{i=1}^{d} y_i X^{i-1}$$

## Fact

$$AX = XC_{P_{min}^{A,v}} \qquad C_{P_{min}^{A,v}} = \begin{pmatrix} 0 & & & m_0 \\ 1 & 0 & & m_1 \\ & \ddots & \ddots & \vdots \\ & & 1 & m_{d-1} \end{pmatrix}$$

*If* $(d = n)$ $\Rightarrow$*one gets* $P_{min}^{A,v}$ *from* $C_{P_{min}^{A,v}} = X^{-1}AX$ *[Krylov]*

# Krylov's method

## Definition

$$K = [\underbrace{v|Av|\dots|A^d v|}_{X} A^{d+1}v|\dots|A^n v]$$

$$\Rightarrow P_{\min}^{A,v}(X) = X^d - \sum_{i=1}^{d} y_i X^{i-1}$$

## Fact

$$AX = XC_{P_{min}^{A,v}} \qquad C_{P_{min}^{A,v}} = \begin{pmatrix} 0 & & & m_0 \\ 1 & 0 & & m_1 \\ & \ddots & \ddots & \vdots \\ & & 1 & m_{d-1} \end{pmatrix}$$

*If* $(d = n)$ $\Rightarrow$*one gets* $P_{min}^{A,v}$ *from* $C_{P_{min}^{A,v}} = X^{-1}AX$ *[Krylov]*

*If* $(d < n)$ *complete* $X$ *into* $\overline{X}$ *invertible*

$$\Rightarrow \overline{X}^{-1} A \overline{X} = \begin{pmatrix} C_{P_{min}^{A,v}} & * \\ 0 & B \end{pmatrix}$$

# Sommaire

# Using LUP factorization

## Completion of $X$

Easier on the triangularized form (LUP) :

# Using LUP factorization

## Completion of $X$

Easier on the triangularized form (LUP) :

Compute $\begin{pmatrix} L_1 & 0 \\ L_2 & Id \end{pmatrix} \begin{pmatrix} U_1 & U_2 \\ 0 & 0 \end{pmatrix} P = X^T$

# Using LUP factorization

## Completion of $X$

Easier on the triangularized form (LUP) :

Compute $\begin{pmatrix} L_1 & 0 \\ 0 & Id \end{pmatrix} \begin{pmatrix} U_1 & U_2 \\ 0 & Id \end{pmatrix} P = \overline{X}^T$

# Using LUP factorization

## Completion of $X$

Easier on the triangularized form (LUP) :

Compute $\begin{pmatrix} L_1 & 0 \\ 0 & Id \end{pmatrix} \begin{pmatrix} U_1 & U_2 \\ 0 & Id \end{pmatrix} P = \overline{X}^T$

$$\Rightarrow \boxed{B = A'_{22} - A'_{21} U_1^{-1} U_2} \text{ where } A' = \begin{pmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{pmatrix} = PA^T P^T.$$

# Using LUP factorization

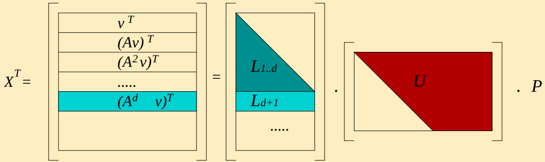
## Completion of $X$

Easier on the triangularized form (LUP) :

$$\text{Compute } \begin{pmatrix} L_1 & 0 \\ 0 & Id \end{pmatrix} \begin{pmatrix} U_1 & U_2 \\ 0 & Id \end{pmatrix} P = \overline{X}^T$$

$$\Rightarrow \boxed{B = A'_{22} - A'_{21} U_1^{-1} U_2} \text{ where } A' = \begin{pmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{pmatrix} = PA^T P^T.$$

## Minimal Polynomial $P_{\mathtt{min}}^{A,v}(X) = X^d - \sum_{i=1}^d y_i X^{i-1}$



# Using LUP factorization

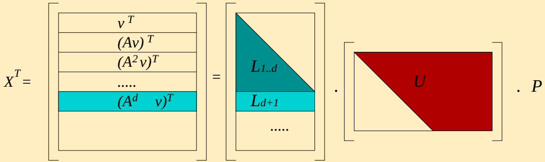
## Completion of $X$

Easier on the triangularized form (LUP) :

$$\text{Compute } \begin{pmatrix} L_1 & 0 \\ 0 & Id \end{pmatrix} \begin{pmatrix} U_1 & U_2 \\ 0 & Id \end{pmatrix} P = \overline{X}^T$$

$$\Rightarrow \boxed{B = A'_{22} - A'_{21} U_1^{-1} U_2} \text{ where } A' = \begin{pmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{pmatrix} = PA^T P^T.$$

## Minimal Polynomial $P^{A,v}_{\min}(X) = X^d - \sum_{i=1}^{d} y_i X^{i-1}$



$$\Rightarrow \boxed{y = L_{d+1} L_{1\dots d}^{-1}} \text{ (in only } \mathcal{O}(n^2) \text{ !)}$$

# LU-Krylov algorithm : `LUK`

```
Require: $A$ a $n \times n$ matrix over a field
Ensure: $P^A_{\text{char}}(X)$ the characteristic polynomial of $A$
```

# LU-Krylov algorithm : `LUK`

```
Require: $A$ a $n \times n$ matrix over a field
Ensure: $P_{\text{char}}^{A}(X)$ the characteristic polynomial of $A$
1: Pick a random vector $v$
2: Compute $X = [v|Av|A^2v|\dots|A^nv]$
```

# LU-Krylov algorithm : `LUK`

```
Require: $A$ a $n \times n$ matrix over a field
Ensure: $P_{\text{char}}^{A}(X)$ the characteristic polynomial of $A$
Pick a random vector $v$
Compute $X = [v|Av|A^2v|\dots|A^nv]$
Compute $(L, U, P) = \texttt{LUP}(X^T)$ $(d = rank(X^T))$
```

# LU-Krylov algorithm : `LUK`

```
Require: $A$ a $n \times n$ matrix over a field
Ensure: $P^A_{\mathrm{char}}(X)$ the characteristic polynomial of $A$
Pick a random vector $v$
Compute $X = [v|Av|A^2v|\ldots|A^nv]$
Compute $(L, U, P) = \mathtt{LUP}(X^T)$ $(d = rank(X^T))$
Solve $y^T L_{1\ldots d} = L_{d+1}$
Set $P^{A,v}_{min}(X) = X^d - \sum_{i=1}^{d} y_i X^{i-1}$
```

# LU-Krylov algorithm : `LUK`

```
Require: $A$ a $n \times n$ matrix over a field
Ensure: $P^A_{\text{char}}(X)$ the characteristic polynomial of $A$
 1: Pick a random vector $v$
 2: Compute $X = [v|Av|A^2v|\dots|A^nv]$
 3: Compute $(L, U, P) = \texttt{LUP}(X^T)$ $(d = rank(X^T))$
 4: Solve $y^T L_{1\dots d} = L_{d+1}$
 5: Set $P^{A,v}_{min}(X) = X^d - \sum_{i=1}^{d} y_i X^{i-1}$
 6: if $(d = n)$ then
 7:     return $P^A_{\text{char}} = P^{A,v}_{\text{min}}$
 8: else


11: end if
```

# LU-Krylov algorithm : `LUK`

```
Require: $A$ a $n \times n$ matrix over a field
Ensure: $P^A_{\text{char}}(X)$ the characteristic polynomial of $A$
Pick a random vector $v$
Compute $X = [v|Av|A^2v|\ldots|A^nv]$
Compute $(L, U, P) = \mathtt{LUP}(X^T)$ $(d = rank(X^T))$
Solve $y^T L_{1\ldots d} = L_{d+1}$
Set $P^{A,v}_{min}(X) = X^d - \sum_{i=1}^{d} y_i X^{i-1}$
if $(d = n)$ then
    return $P^A_{\text{char}} = P^{A,v}_{\text{min}}$
else
    $A' = PA^TP^T$; $B = A'_{22} - A'_{21}S_1^{-1}S_2$

end if
```

# LU-Krylov algorithm : `LUK`

```
Require: $A$ a $n \times n$ matrix over a field
Ensure: $P^A_{\text{char}}(X)$ the characteristic polynomial of $A$
Pick a random vector $v$
Compute $X = [v|Av|A^2v|\dots|A^nv]$
Compute $(L, U, P) = \texttt{LUP}(X^T)$ $(d = rank(X^T))$
Solve $y^T L_{1\dots d} = L_{d+1}$
Set $P^{A,v}_{min}(X) = X^d - \sum_{i=1}^{d} y_i X^{i-1}$
if $(d = n)$ then
    return $P^A_{\text{char}} = P^{A,v}_{\text{min}}$
else
    $A' = PA^TP^T$; $B = A'_{22} - A'_{21}S_1^{-1}S_2$
    return $P^A_{\text{char}}(X) = P^{A,v}_{\text{min}}(X) \times \texttt{LUK}(B)$
end if
```

# LU-Krylov algorithm : `LUK`

```
Require: $A$ a $n \times n$ matrix over a field
Ensure: $P^A_{\text{char}}(X)$ the characteristic polynomial of $A$
Pick a random vector $v$
Compute $X = [v|Av|A^2v|\dots|A^nv]$
Compute $(L, U, P) = \texttt{LUP}(X^T)$ $(d = rank(X^T))$
Solve $y^T L_{1\dots d} = L_{d+1}$
Set $P^{A,v}_{min}(X) = X^d - \sum_{i=1}^{d} y_i X^{i-1}$
if $(d = n)$ then
    return $P^A_{\text{char}} = P^{A,v}_{\text{min}}$
else
    $A' = PA^TP^T$; $B = A'_{22} - A'_{21} S_1^{-1} S_2$
    return $P^A_{\text{char}}(X) = P^{A,v}_{\text{min}}(X) \times \texttt{LUK}(B)$
end if
```

- Deterministic (although based on a probabilistic minpoly !)
- Intensive use of matrix product...
- ... but only $2.666n^3$ algebraic complexity

# Sommaire

# Comparison with the branching algorithm

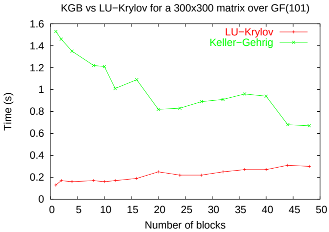
## The branching algorithm

- Also based on Krylov iterates and elimination
- Handles every blocks at once with matrix product
- The best in theory : $\mathcal{O}(n^\omega \log(n))\ \ (= \mathcal{O}(n^\omega \log(k_{\max})))$

# Comparison with the branching algorithm

## The branching algorithm

- Also based on Krylov iterates and elimination
- Handles every blocks at once with matrix product
- The best in theory : $\mathcal{O}(n^\omega \log(n)) \ \ (= \mathcal{O}(n^\omega \log(k_{\max})))$



- Comparison with different number of blocks
- the *log* factor and the constant penalize the gain of grouping operations into matrix product

# Comparison with the fast algorithm

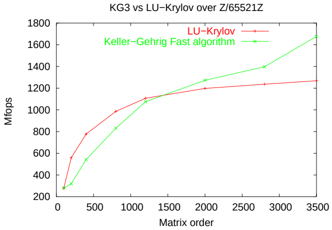
## The Fast algorithm

- Only valid for generic matrices
- Optimal complexity $T = \mathcal{O}(n^\omega)$
- Constant 2.794 with $\omega = 3$ close to the 2.666 of `LUK`

# Comparison with the fast algorithm

## The Fast algorithm

- Only valid for generic matrices
- Optimal complexity $T = \mathcal{O}(n^\omega)$
- Constant 2.794 with $\omega = 3$ close to the 2.666 of `LUK`

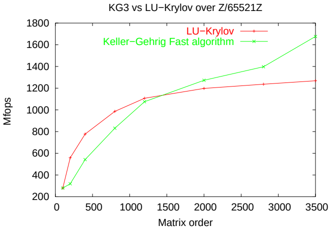


- slower for small matrices (constant)
- faster for large matrices (only matrix products)

# Comparison with the fast algorithm

## The Fast algorithm

- Only valid for generic matrices
- Optimal complexity $T = \mathcal{O}(n^\omega)$
- Constant 2.794 with $\omega = 3$ close to the 2.666 of `LUK`



- slower for small matrices (constant)
- faster for large matrices (only matrix products)

$\Rightarrow$advocates for a generalization or hybrid algorithm.

# Sommaire

# Generalities

Several approaches:

Ring operations

- without divisions *[Berkowitz84,Kaltofen92]*
- with exact divisions *[Abdeljaoued-Malaschonock01]*

# Generalities

Several approaches:

Ring operations

- without divisions *[Berkowitz84,Kaltofen92]*
- with exact divisions *[Abdeljaoued-Malaschonock01]*

Finite fields and chinese remaindering : *Folklore*

# Generalities

Several approaches:

Ring operations

- without divisions *[Berkowitz84,Kaltofen92]*
- with exact divisions *[Abdeljaoued-Malaschonock01]*

Finite fields and chinese remaindering : *Folklore*

Lifting and gcd free basis: *[Storjohann00]*

# Generalities

Several approaches:

Ring operations

- without divisions *[Berkowitz84,Kaltofen92]*
- with exact divisions *[Abdeljaoued-Malaschonock01]*

Finite fields and chinese remaindering : *Folklore*

Lifting and gcd free basis: *[Storjohann00]*

Combination Block-Wiedemann+BSGS *[Kaltofen-Villard04]*

# Sommaire

# Bound on the coefficients

### Chinese Remainder Algorithm

**Principle** Several computations modulo random word size primes $p_i$

**Correctness** If $\beta$ is bounds the result: correctness if $\Pi_i p_i \leq \beta$

# Bound on the coefficients

**Chinese Remainder Algorithm**

**Principle** Several computations modulo random word size primes $p_i$

**Correctness** If $\beta$ is bounds the result: correctness if $\Pi_i p_i \leq \beta$

- for `det` : use Hadamard's bound $\Rightarrow \log(|d|) \leq \frac{n}{2}(\log(n) + \log(\|A\|^2))$

# Bound on the coefficients

## Chinese Remainder Algorithm

**Principle** Several computations modulo random word size primes $p_i$

**Correctness** If $\beta$ is bounds the result: correctness if $\Pi_i p_i \leq \beta$

- for `det` : use Hadamard's bound
  $\Rightarrow \log(|d|) \leq \frac{n}{2}(\log(n) + \log(\|A\|^2))$
- for `charpoly` :
  $\beta = max_{i=0..\frac{\sqrt{1+4en}-1}{2e}} \left( \binom{n}{i} \sqrt{(n-i)\log(\|A|^2)}^{n-i} \right)$
  $\Rightarrow \log(|c_i|) \leq \frac{n}{2}(\log(n) + \log(\|A\|^2) + 0.21163175)$

# Example

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 \end{bmatrix}$$

$$\Rightarrow P^{A}_{\mathtt{char}}(X) = X^5 - 5X^4 + 40X^2 - 80X + 48$$

# Example

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 \end{bmatrix}$$

$$\Rightarrow P^A_{\mathtt{char}}(X) = X^5 - 5X^4 + 40X^2 - 80X + 48$$

- $max_i(|c_i|) = 80 = \binom{5}{1}\sqrt{4}^4$

# Example

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 \end{bmatrix}$$

$$\Rightarrow P^A_{\mathtt{char}}(X) = X^5 - 5X^4 + 40X^2 - 80X + 48$$

- $max_i(|c_i|) = 80 = \binom{5}{1}\sqrt{4}^4$
- Hadamard's bound : 55.9

# Example

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 \end{bmatrix}$$

$$\Rightarrow P^A_{\mathtt{char}}(X) = X^5 - 5X^4 + 40X^2 - 80X + 48$$

- $max_i(|c_i|) = 80 = \binom{5}{1}\sqrt{4}^4$
- Hadamard's bound : 55.9
- bound in [Giesbrecht-Storjohann02] : 21792.7

# Example

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 \end{bmatrix}$$

$$\Rightarrow P^A_{\mathtt{char}}(X) = X^5 - 5X^4 + 40X^2 - 80X + 48$$

- $max_i(|c_i|) = 80 = \binom{5}{1}\sqrt{4}^4$
- Hadamard's bound : 55.9
- bound in [Giesbrecht-Storjohann02] : 21792.7
- Our bound : 80.66661.

# Sommaire

# Early termination

- Stop as soon as the reconstructed polynomial remains unchanged
- Probability of failure $< 1/p$ (rough majoration)

# Early termination

- Stop as soon as the reconstructed polynomial remains unchanged
- Probability of failure $< 1/p$ (rough majoration)

Improvement :

**Algorithm `CIA`**

- CRA on the minimal polynomial (stops earlier)
- recovery of the characteristic polynomial by :
    - factorization of $P^A_{\mathtt{min}}$ over $\mathbb{Z}$ via Hensel Lifting
    - one computation of the characteristic polynomial mod $p$
    - recovery of the multiplicities by divisions in $\mathbb{Z}_p[X]$

# Properties of `CIA`

## Properties

- Las Vegas if `minpoly` is deterministic

  $\Rightarrow$test if $\sum_i \alpha_i d_i = n$

# Properties of `CIA`

### Properties

- Las Vegas if `minpoly` is deterministic

  $\Rightarrow$test if $\sum_i \alpha_i d_i = n$
- Otherwise MonteCarlo with many failure detections

  $\Rightarrow$test if $\alpha_i = 0$ implies $P^A_{\texttt{min}} \nmid P^A_{\texttt{char}}$ mod $p$

  $\Rightarrow$test if $Trace(A) = a_{n-1}$ implies $P^A_{\texttt{min}} \nmid P^A_{\texttt{char}}$

# Properties of `CIA`

## Properties

- Las Vegas if `minpoly` is deterministic

  $\Rightarrow$test if $\sum_i \alpha_i d_i = n$
- Otherwise MonteCarlo with many failure detections

  $\Rightarrow$test if $\alpha_i = 0$ implies $P^A_{\texttt{min}} \nmid P^A_{\texttt{char}}$ mod $p$

  $\Rightarrow$test if $\mathit{Trace}(A) = a_{n-1}$ implies $P^A_{\texttt{min}} \nmid P^A_{\texttt{char}}$
- Also adapted to sparse computations (using Wiedemann `minpoly`), although it requires one dense modular computation

# Sommaire

# Experiments with random dense matrices

On an athlon-1.8Ghz with 2Gb of RAM

| $n$ | |
|---|---|
| 100 | |
| 200 | |
| | |
| 400 | |
| | |
| 800 | |
| | |
| 1200 | |
| | |
| 1500 | |
| | |
| 3000 | |
| | |

# Experiments with random dense matrices

On an athlon-1.8Ghz with 2Gb of RAM

| *n* | `Maple` |
|---:|---:|
| 100 | 163s |
| 200 | 3355s |
| | |
| 400 | 74970s |
| | |
| 800 | |
| | |
| 1200 | |
| | |
| 1500 | |
| | |
| 3000 | |
| | |

# Experiments with random dense matrices

On an athlon-1.8Ghz with 2Gb of RAM

| $n$ | `Maple` | `Magma` |
|---|---|---|
| 100 | 163s | 0.34s |
| 200 | 3355s | 4.45s |
| | | *11.1Mb* |
| 400 | 74970s | 69.8s |
| | | *56Mb* |
| 800 | | 1546s |
| | | *403Mb* |
| 1200 | | 8851s |
| | | *1368Mb* |
| 1500 | | MT |
| | | |
| 3000 | | MT |
| | | |

# Experiments with random dense matrices

On an athlon-1.8Ghz with 2Gb of RAM

| *n* | `Maple` | `Magma` | `ILUK-det` |
|---:|---:|---:|---:|
| 100 | 163s | 0.34s | 0.23s |
| 200 | 3355s | 4.45s | 3.95s |
| | | *11.1Mb* | *3.5Mb* |
| 400 | 74970s | 69.8s | 91.4s |
| | | *56Mb* | *10.1Mb* |
| 800 | | 1546s | 1409s |
| | | *403Mb* | *36.3Mb* |
| 1200 | | 8851s | 7565s |
| | | *1368Mb* | *81Mb* |
| 1500 | | MT | 21010s |
| | | | *136Mb* |
| 3000 | | MT | 349494s |
| | | | *521Mb* |

# Experiments with random dense matrices

On an athlon-1.8Ghz with 2Gb of RAM

| $n$ | `Maple` | `Magma` | `ILUK-det` | `CIA` |
|---|---|---|---|---|
| 100 | 163s | 0.34s | 0.23s | 0.20s |
| 200 | 3355s | 4.45s | 3.95s | 3.25s |
| | | *11.1Mb* | *3.5Mb* | *3.5Mb* |
| 400 | 74970s | 69.8s | 91.4s | 71.74s |
| | | *56Mb* | *10.1Mb* | *10.1Mb* |
| 800 | | 1546s | 1409s | 1110s |
| | | *403Mb* | *36.3Mb* | *36.3Mb* |
| 1200 | | 8851s | 7565s | 5999s |
| | | *1368Mb* | *81Mb* | *81Mb* |
| 1500 | | MT | 21010s | 16705s |
| | | | *136Mb* | *136Mb* |
| 3000 | | MT | 349494s | 286466s |
| | | | *521Mb* | *521Mb* |

# Experiments with structured matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | |
|---|---|
| $n$ | |
| $d$ | |
| $\omega$ | |
| `Magma-prob` | |
| `Magma-det` | |
| `ILUK-det` | |
| `CIA-sparse` | |
| `CIA-dense` | |

Computation time in seconds.

# Experiments with structured matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | $A$ |
|---|---|
| $n$ | 300 |
| $d$ | 75 |
| $\omega$ | 1.9 |
| `Magma-prob` | 1.14 |
| `Magma-det` | 1.31 |
| `ILUK-det` | 1.1 |
| `CIA-sparse` | 0.32 |
| `CIA-dense` | 1.22 |

Computation time in seconds.

# Experiments with structured matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | $A$ | $U^{-1}AU$ |
|---|---|---|
| $n$ | 300 | 300 |
| $d$ | 75 | 75 |
| $\omega$ | 1.9 | 300 |
| `Magma-prob` | 1.14 | 7.11 |
| `Magma-det` | 1.31 | 10.55 |
| `ILUK-det` | 1.1 | 93.3 |
| `CIA-sparse` | 0.32 | 4.32 |
| `CIA-dense` | 1.22 | 1.3 |

Computation time in seconds.

# Experiments with structured matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | $A$ | $U^{-1}AU$ | $A^TA$ |
|---|---|---|---|
| $n$ | 300 | 300 | 300 |
| $d$ | 75 | 75 | 21 |
| $\omega$ | 1.9 | 300 | 2.95 |
| `Magma-prob` | 1.14 | 7.11 | 0.23 |
| `Magma-det` | 1.31 | 10.55 | 0.24 |
| `ILUK-det` | 1.1 | 93.3 | 64.87 |
| `CIA-sparse` | 0.32 | 4.32 | 0.81 |
| `CIA-dense` | 1.22 | 1.3 | 0.87 |

Computation time in seconds.

# Experiments with structured matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | $A$ | $U^{-1}AU$ | $A^TA$ | B |
|---|---|---|---|---|
| $n$ | 300 | 300 | 300 | 600 |
| $d$ | 75 | 75 | 21 | 424 |
| $\omega$ | 1.9 | 300 | 2.95 | 4 |
| `Magma-prob` | 1.14 | 7.11 | 0.23 | 6.4 |
| `Magma-det` | 1.31 | 10.55 | 0.24 | 6.4 |
| `ILUK-det` | 1.1 | 93.3 | 64.87 | 68.4 |
| `CIA-sparse` | 0.32 | 4.32 | 0.81 | 4.4 |
| `CIA-dense` | 1.22 | 1.3 | 0.87 | 38.9 |

Computation time in seconds.

# Experiments with structured matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | $A$ | $U^{-1}AU$ | $A^TA$ | B | $U^{-1}BU$ |
|---|---|---|---|---|---|
| $n$ | 300 | 300 | 300 | 600 | 600 |
| $d$ | 75 | 75 | 21 | 424 | 424 |
| $\omega$ | 1.9 | 300 | 2.95 | 4 | 600 |
| `Magma-prob` | 1.14 | 7.11 | 0.23 | 6.4 | 184.7 |
| `Magma-det` | 1.31 | 10.55 | 0.24 | 6.4 | 185 |
| `ILUK-det` | 1.1 | 93.3 | 64.87 | 68.4 | 2305 |
| `CIA-sparse` | 0.32 | 4.32 | 0.81 | 4.4 | 352.6 |
| `CIA-dense` | 1.22 | 1.3 | 0.87 | 38.9 | 42.6 |

Computation time in seconds.

# Experiments with structured matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | $A$ | $U^{-1}AU$ | $A^TA$ | B | $U^{-1}BU$ | $B^TB$ |
|---|---|---|---|---|---|---|
| $n$ | 300 | 300 | 300 | 600 | 600 | 600 |
| $d$ | 75 | 75 | 21 | 424 | 424 | 8 |
| $\omega$ | 1.9 | 300 | 2.95 | 4 | 600 | 13 |
| `Magma-prob` | 1.14 | 7.11 | 0.23 | 6.4 | 184.7 | 6.04 |
| `Magma-det` | 1.31 | 10.55 | 0.24 | 6.4 | 185 | 6.07 |
| `ILUK-det` | 1.1 | 93.3 | 64.87 | 68.4 | 2305 | 155.3 |
| `CIA-sparse` | 0.32 | 4.32 | 0.81 | 4.4 | 352.6 | 2.15 |
| `CIA-dense` | 1.22 | 1.3 | 0.87 | 38.9 | 42.6 | 2.57 |

Computation time in seconds.

# Other sparse matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | |
|---|---|
| $n$ | |
| $\omega$ | |
| `magma-prob` | |
| `CIA-sparse` | |
| `CIA-dense` | |

Computation time in seconds.

# Other sparse matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | `TF12` |
|---|---|
| $n$ | 552 |
| $\omega$ | 7.6 |
| `magma-prob` | 10.12s |
| `CIA-sparse` | 6.8s |
| `CIA-dense` | 61.77s |

Computation time in seconds.

# Other sparse matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | `TF12` | `Tref500` |
|---|---|---|
| $n$ | 552 | 500 |
| $\omega$ | 7.6 | 16.9 |
| `magma-prob` | 10.12s | 112s |
| `CIA-sparse` | 6.8s | 65.14s |
| `CIA-dense` | 61.77s | 372.6s |

Computation time in seconds.

# Other sparse matrices

On a athlon-1.8Ghz with 2Gb of RAM.

| Matrix | `TF12` | `Tref500` | `mk9b3` |
|---|---|---|---|
| $n$ | 552 | 500 | 1260 |
| $\omega$ | 7.6 | 16.9 | 3 |
| `magma-prob` | 10.12s | 112s | 48.4s |
| `CIA-sparse` | 6.8s | 65.14s | 31.25s |
| `CIA-dense` | 61.77s | 372.6s | 433s |

Computation time in seconds.

# Sommaire

# Toward a truly sparse algorithm

Find the multiplicities by different techniques :

- compute some rank($P_i(A)$) where $P_i$ is an irreducible factor of $P_{\min}^A$
- combinatorial search
- sieve remaining solutions by some evaluation of $P_{\text{char}}^A$
- ...

# Toward a truly sparse algorithm

Find the multiplicities by different techniques :

- compute some rank($P_i(A)$) where $P_i$ is an irreducible factor of $P^A_{\text{min}}$
- combinatorial search
- sieve remaining solutions by some evaluation of $P^A_{\text{char}}$
- ...

Applied to a graph theory problem :

⇒compute the characteristic polynomial of a $7140 \times 7140$ sparse adjacency matrix in 1h4' on a P4-2.4Ghz

Thank you !