MATH 895, Course Project, Summer 2013

Instructor: Michael Monagan

The project is worth 25% of your final grade. You should expect to spend about twice the time you would spend on an assignment. There are three suggested projects. Each one requires that you read a paper, implement two algorithms, and generate some timing data to verify the claims made in the paper, then write up a report.

You may either write a traditional report in LaTeX or create a poster for presentation at this years *Symposium on Mathematics and Computation* here at SFU on Wednesday August 7th. See

`http://mathcomp2013.irmacs.sfu.ca/`

The main event at this meeting is the poster session where students in our department, both undergraduate and graduate, present their work. I will pay for your registration fee to attend the symposium.

If you choose to write a report, it should be about 5–7 pages (12pt font). Additionally, include an Appendix containing Maple code or Maple worksheets with any data that you want to include.

If you do a poster, you can use my LaTeX poster outline on the course website as an outline. I will pay for the cost of printing your poster. And, to encourage you, I will give you one extra grade point (5%).

I anticipate your spending about 50% of your time on the content (implementing the algorithm(s) and generating the timing data) and 50% of your time reading the paper and writing the report or creating and presenting the poster.

LaTeX: I will need to give you a tutorial on using LaTeX. Most of what you need you can copy from my LaTeX file for this document. See the file project.tex. Also helpful: the Maple command `latex(f);` will generate LaTeX for a mathematical formula f (including matrices).

Project 1: Determinant Algorithms

Read the paper *Analysis of Algorithms, A Case Study: Determinants of Matrices with Polynomial Entries* by Gentleman and Johnson. It is posted on the course website. It compares the efficiency of fraction-free Gaussian elimination with minor expansion on matrices with *polynomial* entries.

Program two Maple procedures `FracFree(A,n)` and `MinorExp(A,n)` which both compute $\det(A)$ for an n by n matrix A of polynomials in $\mathbb{Q}[x_1, x_2, ..., x_m]$.

`FracFree(A,n)` should use the Bareiss fraction-free Gaussian elimination that you implemented this on Assignment 3 for matrices of integers. For matrices of polynomials use `expand` for polynomial multiplication and `divide(A,B,'Q')` for polynomial division.

`MinorExp(A,n)` should use the method of minor expansion that avoids recomputation of minors as described in the paper. The Maple library routine combinat[choose] may be helpful. The `option remember` facility may also be helpful. Feel free to talk with me about how to do this.

Time your algorithms on the following three types of matrices.

The $n \times n$ symmetric Toeplitz matrix for $(n = 4)$:

$$\begin{bmatrix} w & x & y & z \\ x & w & x & y \\ y & x & w & x \\ z & y & x & w \end{bmatrix}$$

The $n \times n$ cyclic shift matrix for $(n = 4)$:

$$\begin{bmatrix} w & x & y & z \\ z & w & x & y \\ y & z & w & x \\ x & y & z & w \end{bmatrix}$$

$n \times n$ Random univariate matrices:

```
> d := 3:
> n := 4:
> R := proc() randpoly(x,degree=d,dense) end:
> A := Matrix(n,n,R);
```

You should find that the fraction free method is best on the univariate polynomial matrices (where d is fixed by n grows) but the minor expansion method is best on the multivariate benchmarks for all $n > 2$. Thinking about the two algorithms, one significant difference is that minor expansion has no divisions. We say it is division free. To investigate this further, recall that at the end of the fraction free elimination, $A_{n,n} = \pm \det(A)$ and that $A_{n,n}$ is computed as

$$A_{n,n} = \frac{A_{n-1,n-1}A_{n,n} - A_{n,n-1}A_{n-1,n}}{A_{n-2,n-2}}$$

Thus the numerator in this division equals $A_{n,n} \times A_{n-2,n-2}$ and thus must be bigger than $\det(A)$. Print out the number of terms of this numerator and $\det(A)$. Generate a table (for each benchmark) of this data along with the timings for both algorithms. Your table should look something like this

n	# det A	max terms	time (FracFree)	time (MinorExp)
...				
6	120	575	0.002s	0.003s
7	427	3277	0.008s	0.008s
8	1628	21016	0.058s	0.053s
9	6090	128530	0.553s	0.087s
...				

Table 1: Maple 17 timings (in seconds) for n by n symmetric Toeplitz matrices.

Project 2: Polynomial Division Algorithms

Read the paper *Polynomial division using dynamic arrays, heaps and packed exponent vectors* by Monagan and Pearce. It is posted on the course website. It compares the efficiency of polynomial division algorithms using arrays, heaps and geo-buckets. The goal of this project is to implement and compare the two heap based division algorithms for multivariate polynomial division with the naive merging algorithm.

For polynomials in $\mathbb{Q}[x_1, x_2, ..., x_n]$ program three boolean Maple procedures

1 `QuoHeapDivide(A,B,n,'Q')`,

2 `DivHeapDivide(A,B,n,'Q')` and

3 `MergeDivide(A,B,n,'Q')`.

that test if B divides A in $\mathbb{Q}[x_1, x_2, ..., x_n]$ and if it does, assigns Q the quotient of $A \div B$. The parameter n is the number of variables. You can represent the polynomial as a list of terms in the form [c,m] where c is a coefficient and m is an exponent vector or the encoding of an exponent vector. The terms should be sorted (in descending order) in the graded lexicographical order. For example, the polynomial $3x^2y + 5xy^2 - 7y^4 \in \mathbb{Q}[x, y]$ could be represented as

$$[\,[-7, [0, 4]],\quad [3, [2, 1]],\quad [5, [1, 2]]\,]$$

Now, letting $Q_1, Q_2, Q_3, \ldots$ denote the terms of the quotient, the `MergeDivide` algorithm should implement the naive division algorithm using $(((A - Q_1B) - Q_2B) - Q_3B) - \ldots$ where you subtract using merging. We know this is inefficient for sparse polynomials.

The `QuoHeapDivide(A,B,n,'Q')` procedure should implement Johnson's division algorithm that you implemented for Assignment 5. Here the product QB is computed using a heap on the terms of the quotient, i.e., using

$$A - \sum_{i=1}^{\#Q} Q_i B.$$

The `DivHeapDivide(A,B,n,'Q')` procedure should implement Monagan and Pearce's divisor heap algorithm where the product QB is computed using a heap on the terms of the divisor B, i.e.,

$$A - \sum_{i=1}^{\#B} B_i Q.$$

Time your algorithms on the following polynomial benchmarks. Here I've shown the division using Maple.

Dense benchmark

```
> d := 16;
> for n from 2 to d-2 do
>     F := expand( (1+x)^d*(1+y)^d*(1+z)^d );
>     B := expand( (1-x)^d*(1-y)^d*(1-z)^d );
>     A := expand( F*B );
>     st := time(); divide(A,B,'Q'); tt := time()-st;
> od:
```

Fateman benchmark

```
> d := 24;
> for n from 3 to d-3 do
>     F := expand( (1+x+y+z)^n );
>     B := expand( (1+x+y+z)^(d-n) );
>     A := expand( F*B );
>     st := time(); divide(A,B,'Q'); tt := time()-st;
> od:
```

Sparse benchmark

```
> d := 36;
> for n from 2 to d-2 by 4 do
>     F := expand( (1+x)^d*(1+y)^d );
>     B := expand( (1+x)^(n-d)*(1+z)^(n-d) );
>     A := expand( F*B );
>     st := time(); divide(A,B,'Q'); tt := time()-st;
> od:
```

Now time your algorithm on these benchmarks. Compute also the number of monomial comparisons that each algorithm takes. Generate a table (for each benchmark) of this data along with the timings for both algorithms. Include the number of terms on A, B and Q in the table so that the table looks like

				MergeDivide		QuoHeapDivide		DivHeapDivide	
n	$\#A$	$\#B$	$\#Q$	time	#comp	time	#comp	time	#comp
2	3885	1225	9	?	?	0.663	70267	?	?
6	8029	961	49	?	?	3.678	465989	?	?
10	10989	729	121	?	?	7.492	954503	?	?

Table 2: Maple 17 timings (in seconds) for the Sparse benchmark.

Project 3: Polynomial GCD Algorithms

Read the 1979 paper *Probabilistic Algorithms for Sparse Polynomials* by Zippel. It is posted on the course website. It develops a sparse GCD algorithm and compares it with the efficiency of several GCD algorithms, including Brown's dense algorithm (see column Modular) on page 224.

Let A, B be polynomials in $\mathbb{Z}[x_1, x_2, ..., x_n]$ and let $G = gcd(A, B)$. For your MGCD and PGCD algorithms from assignment 2, modify both of them to use Zippel's sparse interpolation. Call these new procedures SparseMGCD and SparsePGCD. I suggest you pass the variables as a list as a parameter so that you code `SparseMGCD(A,B,X)` and `SparsePGCD(A,B,X,p)` for prime p. To simplify the coding of PGCD assume that G is monic in x_1 i.e. $G = cx_1^d + f(x_1, x_2, ..., x_n)$ where $\deg_{x_1} f < d$ and $c \in \mathbb{Z}$.

Let

$$G = \sum_{i=1}^{m} c_i(x_2, ..., x_n)x_1^i \quad \text{where} \quad c_m \in \mathbb{Z}.$$

Suppose t is the maximum number of terms in the coefficients of G, i.e. $t = \max_{i=1}^{m-1} \#c$. When you implement the algorithm you will construct systems of linear equations, one for each coefficient in x_1 of G, the biggest of which will be $t \times t$. Because we use random evaluation points modulo p, it is possible that a linear system will be underdetermined, i.e., have rank $< t$. If this happens you need another equation. It is also possible that the "skeleton" of the previous result from PGCD is wrong and we need some way to detect this. Use one more univariate image (so $t+1$ images) so that you have one more equation than the number of unknowns to detect this case. If the skeleton is wrong then the $t+1$ by t linear system will be inconsistent with high probability.

In the appendix of the paper you will see 10 test problems. In the test problems the inputs are $d_i f_i$ and $d_i g_i$ where the gcd is the d_i polynomials. But they are not monic so make them monic in x_1 by adding x_1^m for $m = 1 + \deg_{x_1} d_i$. Zippel reports timings only for the benchmarks. It would be more helpful if we counted U, the number of univariate gcds in $\mathbb{Z}[x_1]$ that PGCD makes. Use the prime $p = 2^{31} - 1$ so that one prime is sufficient for MGCD. Generate a table of timings and U for procedures MGCD and SparseMGCD. For each benchmark include the number variables n and the maximum number terms t and the number of univariate GCDs computed U. So your table will look something like this

			Brown		Zippel	
Test	n	t	time	U	time	U
1	1	1	?	?	?	?
2	2	2	?	?	?	?
3	3	2	?	?	?	?
4	4	3	?	?	?	?
...						

Table 3: Maple 17 timings (in seconds) for the Zippel's benchmarks.

Zippel's benchmarks all have very small values for t which makes his algorithm look better than it performs on real problems that occur in practice. His method needs to solve linear systems of size $t+1 \times t+1$ which costs $O(t^3)$ time and $O(t^2)$ space which is bad for large t. Zippel's 1990 paper *Interpolating Polynomials from their Values* (also on the course webpage) shows how to choose the evaluation points in such a way that he can solve the linear systems in $O(t^2)$ time and $O(t)$ space – if you are interested. Anyway, include this benchmark as an 11'th benchmark with $t = 21$.

$$d_{11} = (x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 6x_6 + 7)^2 + 3, f_{11} = d_{11} + x_6, g_{11} = d_{11} + x_3.$$

Discuss whether the number of univariate GCDs agree with the theory or not. SparsePGCD should use at most $(n-1)(d+1)(t+1)$ univariate GCDs where $d = \max_{i=2}^{n} \deg_{x_i} G$. PGCD should use at most $\Pi_{i=2}^{n}(1 + \deg_{x_i} G)$.