MATH 895, Course Project, Summer 2015

Instructor: Michael Monagan

The project is worth 25% of your final grade. You should expect to spend about 2.5 times the time you would spend on an assignment. There are three suggested projects. Each one requires that you read two papers, implement an algorithm, think about something, then present your work.

To present your work you may either write a report in LaTeX or create a poster for presentation at this years *Symposium on Mathematics and Computation* here at SFU on Thursday August 6th. See

`http://mathcompsymposium.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 do the poster option then the project is effectively due August 6th.

If you choose to write a report, it should be about 5–10 pages (12pt font). Additionally, you may include an Appendix containing Maple code or Maple worksheets with any data that you wish to include. **The report is due Thursday August 20th.**

If you choose to 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.

LaTeX: 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 and 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.

Read the paper Lazy and Forgetful Polynomial Arithmetic and Applications by Paul Vrbik and Michael Monagan on the course website. The authors avoid creating the polynomial $A_{n-1,n-1}A_{n,n} - A_{n,n-1}A_{n-1,n}$ explicitly. Explain using pseudo-code with comments how you would to compute

$$det(A) = \pm\frac{A_{n-1,n-1}A_{n,n} - A_{n,n-1}A_{n-1,n}}{A_{n-2,n-2}}$$

without explicitly constructing the numerator. Hint: use heaps to do the multiplications and division. The paper Lazy and Forgetful Polynomial Arithmetic and Applications by Paul Vrbik and Michael Monagan on the course website gives a general solution to this problem. The problem of not expanding the numerator $A_{n-1,n-1}A_{n,n} - A_{n,n-1}A_{n-1,n}$ explicitly is was the motivation for the paper.

Project 2: Gcds over Algebraic Number Fields

Read the paper Computing GCDs of Polynomials over Algebraic Number Fields by Encarnacion and the paper A Modular GCD algorithm over Number Fields presented with Multiple Extensions by Monagan and van Hoeij which are posted on the course website.

For polynomials a, b in $\mathbb{Q}(\alpha)[x]$ implement Encarnacion's algorithm to compute the monic gcd g of a and b using rational reconstruction. Let $m(z) \in \mathbb{Z}[z]$ be the minimal polynomial for α. You will need to run the monic Euclidean algorithm to compute $gcd(a, b) \bmod p$ so over the finite ring $R = \mathbb{Z}_p[z]/m(z)$ where there may be zero divisors. You can do this using the RootOf representation modulo a prime p i.e. using

```
> m := z^2+z+1;
> alias( alpha=RootOf(m,z) );
> g := x^2+1/2*alpha*x+1;
> a := evala(Expand( g*(x^2+alpha*x+1/3) ) );
> b := evala(Expand( g*(x^2-2*alpha*x+1) ) );
> gcd(a,b); # Maple's implementation
                                2
                               x  + 1/2 alpha x + 1

> p := 11;
> Gcd(a,b) mod p;
                                          2
                               6 alpha x + x  + 1
```

In the paper by van Hoeij and Monagan, a prime for which the monic Euclidean algorithm encounters a zero divisor in R is called a "fail prime". Their solution is to simply skip that prime. If Maple code encounters a zero divisor you will get an error which you can "trap" and move on to the next prime as shown below.

```
> p := 13;
> Factor(m) mod p;
                                  (z + 4) (z + 10)

> Gcd( (alpha+4)*x+1, (alpha+10)*x+2 ) mod 13;
Error, (in evalgf1/Gcd) the modular inverse does not exist
> g := traperror( Gcd( (alpha+4)*x+1, (alpha+10)*x+2 ) mod 13 );

                        g := "the modular inverse does not exist"
```

Test your algorithm on your own examples, including $m(z) = z^4 + z^3 + z^2 + z + 1$. Now suppose $m(z) \in \mathbb{Z}[z]$ satisfies $m(\alpha) = 0$, $\deg_z(m) > 0$ but $m(z)$ is reducible over $\mathbb{Q}$. Such applications occur in solving polynomial systems of equations and we'd still like to compute $\gcd(a, b)$ if it exists. Modify the modular GCD algorithm to have the following properties.

> If the monic Euclidean algorithm when executed over $\mathbb{Q}$ encounters a zero divisor then the modular GCD algorithm should output a factor of $m(z)$, otherwise it should output the monic $gcd(a, b)$.

For example if $m(z) = (3z-1)(z^2+1)$, on input of $a = x^2 + zx + 2$ and $b = x^2 + 1/3x + 1$ the monic Euclidean algorithm will compute the remainder $r = a \div b = (z-1/3)x+1$. Now to make the remainder monic we need to invert $\mathrm{lc}(r) = z - 1/3$ which is a zero divisor. This will be discovered when attempting to invert $z - 1/3$ modulo $m(z)$ using the extended Euclidean algorithm. So the modular GCD algorithm should reconstruct $z - 1/3$ from modular images.

You will need to implement the monic Euclidean algorithm in $R[x]$ youself so that when you make r_i monic, you can run the extended Euclidean algorithm on $m(z)$ and $\mathrm{lc}(r_i)$ modulo p to compute the inverse of $\mathrm{lc}(r_i)$ it it exists, otherwise you will have a factor of $m(z)$ modulo p. You will need to design the algorithm so you can prove termination; in a finite number of steps it either outputs $g = gcd(a, b)$ or a factor of $m(z)$.

Project 3: Zippel's GCD Algorithm

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.

Zippel's algorithm is probabilistic.
State and prove the Schwarz Zippel Lemma.

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" or "assumed form" 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 2: 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. 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 $\Pi_{i=2}^{n}(1 + \deg_{x_i} G)$.

Zippel's 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. You can implement this if you would like to.