MATH 800 Course Project, Fall 2023
Computing Inverses of Power Series

Michael Monagan

The project is worth 20% of your final grade. Hand in by 11pm Friday December 15th, 2023.

Part A: Classical Methods

To divide a by b using the fast division algorithm we need to compute $1/b^r \bmod x^{1+m}$ where m is the degree of the quotient. This is equivalent to computing the power series for $1/b^r$ to order $O(x^{1+m})$. For example, if $b = 1 - x$ and $m = 3$ then

$$\frac{1}{1-x} \bmod x^4 = 1 + x + x^2 + x^3.$$

Suppose we have a power series $b(x) = \sum_{k=0}^{\infty} b_k x^k$ over a commutative ring R. If a_0 is invertible then the inverse of $b(x)$ exists. To compute $b^{-1} \bmod x^n$ (up to the term of degree $n-1$) we can use series long division; we divide

$$b_0 + b_1 x + \cdots + b_{n-1}x^{n-1} \,\overline{)\,1 + 0x + \cdots + 0x^{n-1}}$$

just like with the normal division algorithm but in reverse; we divide the term of lowest degree in the dividend by b_0 at each step. **Do this for $b(x) = 1 - x - x^2$ for $n = 6$ by hand.**

Series long division is equivalent to the following. Let $b^{-1} \bmod x^n = a(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$. We require $a(x)b(x) \bmod x^n = 1$, that is

$$(a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1})(b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1}x^{n-1}) \bmod x^n = 1.$$

Equating the coefficient of x^k on both sides we obtain the following equations for $a_0, a_1, \ldots, a_{n-1}$

$$\begin{aligned}
a_0 b_0 &= 1 \\
a_0 b_1 + a_1 b_0 &= 0 \\
a_0 b_2 + a_1 b_1 + a_2 b_0 &= 0 \\
&\vdots \\
a_0 b_{n-1} a_1 b_{n-2} + \cdots + a_{n-1} b_0 &= 0.
\end{aligned}$$

Solving these equations for a_0 then a_1 then a_2 etc. leads to a simple algorithm for computing $a(x)$ which is easy to implement using arrays. Let B be the input array $\boxed{b_0}\boxed{b_1}\boxed{\ldots}\boxed{b_{n-1}}$ indexed from 0 and let A be the array $\boxed{a_0}\boxed{a_1}\boxed{\ldots}\boxed{a_{n-1}}$, indexed from 0. Let p be a prime and $R = \mathbb{Z}_p$ be the coefficient ring. Write a Maple procedure that on input of (B, n, p) outputs the array A containing $b^{-1} \bmod x^n$. Test your algorithm on (i) $b = 1 - x - x^2$ for $n = 8$ and $p = 101$ and (ii) $b = 92\,x^7 + 44\,x^6 + 95\,x^5 + 5\,x^4 + 97\,x^3 + 58\,x^2 + 43\,x + 99$ for $n = 8$ and $p = 101$.

How many multiplications does this algorithm do? Give an exact count.

Part B: The Middle Product

The Newton iteration formula for computing the inverse of $b(x) \mod x^n$ is

$$y_k = 2y_{k-1} - by_{k-1}^2 \bmod x^n$$

where $n = 2^k$ and we have already computed y_{k-1} such that $by_{k-1} = 1 \bmod x^{n/2}$. One way to do this is in a simple loop

```
$n = 1$
$y_0 = 1/\text{coeff}(b, x, 0)$
while $n \le m$ do
        $n = 2n$
        $y_k = 2y_{k-1} - (b \bmod x^n)y_{k-1}^2 \bmod x^n$
end while
```

Since $\deg(y_{k-1}, x) < n/2$ we can use an FFT of size n to compute y_{k-1}^2 but for the second multiplication of y_{k-1}^2 by $b \mod x^n$ we need an FFT of size $2n$. So one multiplication of degree $n/2 - 1$ and one of degree $n - 1$, i.e. $M(n/2) + M(n)$. This leads to the following recurrence for $I(n)$ for the cost of the inverse.

$$I(n) = I(n/2) + M(n/2) + M(n).$$

Using the assumption that $2M(n/2) < M(n)$, that is, one multiplication of degree n costs more than two multiplications of degree $n/2$, we obtain $I(n) < 3M(n)$.

Consider the following alternative formula for the Newton iteration.

$$y_k = 2y_{k-1} - b\,y_{k-1}^2 = y_{k-1} + y_{k-1}(1 - b\,y_{k-1}).$$

Since y_{k-1} satisfies $b\,y_{k-1} = 1 \bmod x^{n/2}$, if we truncate $b \bmod x^n$ we have $(b \bmod x^n)y_{k-1}$ equals

$$1 + 0\,x + \cdots + 0\,x^{\frac{n}{2}-1} + m_0\,x^{\frac{n}{2}} + \cdots + m_{\frac{n}{2}-1}x^{n-1} + a_0\,x^n + a_1\,x^{n+1} + \cdots + a_{\frac{n}{2}-2}x^{\frac{3}{2}n-2}$$

for some coefficients m_i and a_i. Let $m(x) = \sum_{i=0}^{n/2-1} m_i x^i$ and let $a(x) = \sum_{i=0}^{n/2-2} a_i x^i$. We have

$$(b \bmod x^n)y_{k-1} = 1 + m(x)x^{\frac{n}{2}} + a(x)x^n.$$

What we want to compute is $m(x)$ because mod x^n the $a(x)$ is not needed. $m(x)$ is called the middle product. The middle product optimization is to compute $m(x)$ by multiplying $b \bmod x^n$ by y_{k-1} using an FFT of size n. The idea is due to Hanrot, Quercia and Zimmermann [1].

Let ω be a primitive n'th root of unity. Let $F_\omega : F^n \to F^n$ denote the Fourier transform $F_\omega(a) = [a(\omega^i) : 0 \le i \le n-1]$. Since $(\omega^i)^n = (\omega^n)^i = 1$ we have

$$F_\omega(a(x)x^n) = F_\omega(a(x)).$$

Thus if we use an FFT of size n to multiply $b \mod x^n$ by y_{k-1} we will get

$$1 + a(x) + m(x)x^{n/2}$$

and we can read off $m(x)$ since $\deg a(x) < n/2$. Note, after inverting the FFT we will have this vector of n values

$1+a_0$	a_1	a_2	$\ldots$	$a_{\frac{n}{2}-2}$	0	m_1	m_2	$\ldots$	$m_{\frac{n}{2}-1}$

from which we can also read off $m(x)$. Back to the main formula.

$$\begin{aligned} y_k &= y_{k-1} + y_{k-1}(1 - b y_{k-1}) \bmod x^n \\ &= y_{k-1} + y_{k-1} m(x) x^{n/2} \bmod x^n \end{aligned}$$

Thus we have one more multiplication of y_{k-1} of degree $< n/2$ by $m(x)$ of degree $< n/2$ for which we can use an FFT of size n. Thus two multiplications of degree $< n/2$ in total, that is, $2M(n/2)$. This leads to a new cost for computing $1/b \bmod x^n$ of

$$I(n) = I(n/2) + 2M(n/2).$$

Assuming $2M(n/2) < M(n)$ **show that** $I(n) < 2M(n)$**.** Using your FFT code and fast multiplication algorithm from Assignment 3, program the new Newton iteration in Maple to use the middle product optimization and the FFT. Test your algorithm on (i) $b = 1 - x - x^2$ for $n = 8$ and $p = 97$ and (ii)

```
> b := Randpoly(15,x) mod 97;
```

for $n = 16$ and $p = 97$. Check that your answer is correct.

References

[1] Guillaume Hanrot, Michel Quercia, Paul Zimmermann. The Middle Product Algorithm I. Speeding up the division and square root of power series. *Applicable Algebra in Engineering, Communication and Computing* **14**(6): 415–438. 2006.