Implementation of various g.c.d. algorithm in Reduce

September 9, 2018

author: Jin Lianyuan

Chapter 1

Introduction

1.1 Background

In algebra, the greatest common divisor (frequently abbreviated as GCD) of two polynomials is a polynomial, of the highest possible degree, that is a factor of both the two original polynomials. This concept is analogous to the greatest common divisor of two integers.i If we get GCD of two polynomials, we can simplify quotient of two polynomials which can accelerate calculation and simplify expression. Besides, we can use g.c.d. of two polynomials to get least common multiple of two polynomials.

1.2 Problem Description

In this project, we are required to implement different algorithm of g.c.d. They are both based on traditional Euclid algorithm. But the details are different and we should compare and analysis them.

Chapter 2

Algorithm Description

2.1 Algorithm Specification

Euclid algorithm is a great algorithm to derive great common divisor of two polynomials. However, polynomial and integer are different. If we use Euclid algorithm directly on polynomials, we probably have to calculate very big numbers. Besides, polynomials may have many variables which make the problem even more difficult. So there are many modified algorithm to delete common divisor of coefficients.

The first algorithm is Primgcd. In this algorithm we calculate primitive polynomial every time we get pseudoremainder. In this method we can guarantee that our coefficients are least every iteration. However, we must calculate gcd every iteration which is really costly. The second method is SR-Euclid's algorithm. We eliminate common divisor by a formula every iteration. The third and fourth algorithms , Hearn basic and Hearn full are similar. We maintain a list and every iteration we test if pseudoremainder can be divided by element in the list. Since these algorithms are post on the slide in the class, I omit some details and pseudo code. The final algorithm we should modify Hearn full to get a better management of list.

Here is my algorithm: in every iteration after we add a new element in the list we maintain the list and we can guarantee that **there are no trivial divisor (0, 1 and -1) and any two elements ai and aj in list,**

neither ai mod aj equals 0 nor aj mod ai equals 0.

Here is my pseudo-code:

Algorithm maintaining(list, element)

```
1   for each x in list
2         if element mod x = 0
3               element = element / x until element mod x != 0
4   add element into an empty queue
5   append element to the list
6   while queue is not empty
6         a := queue[0]              first element in the queue
7         queue := rest queue except a
8         if a != 1 and a != -1
8               for each x in list
9                     if x mod a =0 and x is not a itself
10                          x := x / a until x mod a != 0
11                          append x into the list
12   delete all -1s, 1s and 0s in the list
13   return list
```

2.2 Validity of the algorithm

Here we prove that we archive our goal. First it is easy to prove that there are no trivial elements in list Since in step 12 we delete these elements explicitly. It is harder to prove any two elements ai and aj in list, neither ai mod aj equals 0 nor aj mod ai equals 0. We prove it by mathematical induction. First, if there is only one element in the list, it is true. Second if there are k(k>=1) elements in the list, we add a new element in the list. The principle preserves since at step 1,2 and 3 we can conclude that the new element appended to the list can't be divided by the elements in the list before. If there exist ai and aj such that ai / aj = 0. aj must be in the queue sometime because if aj is a new element, aj is in the queue at the beginning. If aj is a old elements, ai must be a old elements because step 1,2,3 guarantee it. Since the principle preserve in the old list. aj must be changed by someone else and because of step 9. However, it contradicts step 10 because if ai / aj = 0 we must set ai := ai / aj at step 10. In general, we verify that our algorithm is correct.

Chapter 3

Testing Results

3.1 Test Environment

CPU	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70GHz
RAM	8.00GB
OS	Windows 10 64 bit

3.2 Test cases and results

f,g are two polynomial

f := (x+1)^2*x;

g := (x+1)*x^2;

method	average time/ms
Hearnfull	19
Hearnbasic	15
sreuclid	18
primgcd	13
built-in gcd	13
JHDgcd	13

f := x;

g := 2*x^3+3*x;

method	average time/ms
Hearnfull	11
Hearnbasic	10
sreuclid	10
primgcd	9
built-in gcd	10
JHDgcd	9

f := x^8 + x^6 – 3*x^4 – 3*x^3 + 8*x^2 + 2*x – 5;

g := 3*x^6 + 5*x^4 – 4*x^2 – 9*x – 21;

method	average time/ms
Hearnfull	12
Hearnbasic	10
sreuclid	10
primgcd	12
built-in gcd	9
JHDgcd	12

h:=13*x^7+7*x^5+12*x^3+x+17;

f:=4*h*(x+13)^2*(x+3)^4;

g:=6*h*(x-1)^2*(x+3)^2*(x+13)^3;

method	average time/ms
Hearnfull	77
Hearnbasic	75
sreuclid	81
primgcd	75
built-in gcd	70
JHDgcd	72

f:=(x+y+z)*x;

g:=(x+y+z)*x^2;

method	average time/ms
Hearnfull	14
Hearnbasic	14
sreuclid	16
primgcd	19
built-in gcd	14
JHDgcd	18

f:=x^8+x^6-3*x^4-3*x^3+8*x^2+2*x-5;

g:=3*x^6+5*x^4-4*x^2-9*x-21;

f:=f*(y+1);

g:=g*(y+1);

method	average time/ms
Hearnfull	19
Hearnbasic	16
sreuclid	22
primgcd	16
built-in gcd	9
JHDgcd	26

H:=2*(x1+1)*(x2+1)*(x3+1)-3;

F:=H*(x1-2)*(x2-2)*(x3-2);

G:=H*(x1+2)*(x2+2)*(x3+2);

method	average time/ms
Hearnfull	73
Hearnbasic	74
sreuclid	91
primgcd	66
built-in gcd	29
JHDgcd	89

H:=2*(x1+1)*(x2+1)*(x3+1)*(x4+2)*(x5+3)-3;

F:=H*(x1-2)*(x2-2)*(x3-2)*(x4-3)*(x5+7);

G:=H*(x1+2)*(x2+2)*(x3+2)*(x4-2)*(x5-7);

method	average time/ms
Hearnfull	1787
Hearnbasic	1833
sreuclid	2599
primgcd	1489
built-in gcd	185
JHDgcd	2132

U:=x1-x2*x3+1;

V:=x1-x2+3*x3;

F:=U^2*V^4;

G:=U^4*V^2;

method	average time/ms
Hearnfull	352
Hearnbasic	373
sreuclid	403
primgcd	302
built-in gcd	214
JHDgcd	412

Chapter 4

Analysis and Comments

4.1 Complexity

From test we can find that it is hard to figure out which algorithm is best since the speed of the algorithm is based on which test set you select. The time complexity of traditional Euclid algorithm is $\Omega(\log(n))$. However, There are recursion steps in the new algorithm, so the time complexity can be $m\Omega(\log(n))$, m is the number of variables.

4.2 Conclusion

By the test we can find that HearnFull is a really good algorithm and it is also relatively easy to implement. My original algorithm didn't perform so good as I imagined before. I think it is because the process in my algorithm is too complicated and I can't analysis the upper bound of time complexity.

Appendix

Source Code

```
% FINAL CODE
procedure mygcd(a,b);
begin
	scalar r;
	if a < 0 then a := -a;
	if b < 0 then b := -b;
	while b neq 0 do
	begin
		r := a mod b;
		a := b;
		b := r;
	end;
	return a;
end;

procedure CoeffgcdHF(poly,mvar);
begin
	scalar res,ans;
	res := coeff(poly, mvar);
	%write res;
	ans := lcof(poly, mvar);
	foreach x in res do
		ans := HeuclidFull(x,ans);
	return ans;
end;

procedure PrimipolyHF(x, mvar);
begin
	scalar temp;
	temp:=CoeffgcdHF(x, mvar);
	%write temp;
	if temp neq 0 then
	x:=x/temp;
	return x;
```

```
end;




procedure HEuclidFull(a,b);
begin
	scalar r,l,xx,mvar, temp, coe1, coe2;
	if b=0 then return a;
	if a=0 then return b;
	if numberp a and numberp b then return mygcd(a,b);

	if mainvar(b) = 0 then mvar := mainvar(a) else mvar:=mainvar(b);


	if deg(a, mvar) < deg(b, mvar) then
	begin
		temp := a; a := b; b := temp
	end;

	coe1 := CoeffgcdHF(a,mvar);
	coe2 := CoeffgcdHF(b,mvar);
	a := a/coe1;
	b := b/coe2;
	r:=second(pseudo_divide(a,b,mvar));

	l:=list(lcof(a,mvar),lcof(b,mvar),lcof(r,mvar));

	%write r,l;
	while r neq 0 do
	begin
		%write l;
		a:=b;
		b:=r;
		%if mainvar(b) = 0 then mvar := mainvar(a) else mvar:=mainvar(b);
		r :=second(pseudo_divide(a,b,mvar));
		foreach xx in l do
		begin
			if xx neq 0 and abs(xx) neq 1 and r neq 0 and remainder(r,xx,mvar) eq 0 then
			begin
				while remainder(r,xx,mvar) eq 0
				do
				begin
					%write second(divide(r,xx,mvar));
```

```
                    %write r;
                    %write xx;
                    r := r / xx;
                end;
            end;
        end;
        %write r;
        r := PrimipolyHF(r,mvar);
        l := append(l,{lcof(r,mvar)});
    end;
    return primipolyHF(b,mvar)*HEuclidFull(coe1, coe2);
end;



%aa:=x^8 + x^6 - 3*x^4 - 3*x^3 + 8x^2 + 2*x - 5;
%bb:=3x^5 + 5*x^4 - 4*x^2 - 9*x - 21;
%heuclidbasic(aa,bb);
%for iter:=1 step 1 until len do
%begin
%if r mod fisrt(l) eq 0 then l:=rest(l) else l := append(rest(l),fisrt(l));
%end;

procedure mygcd(a,b);
begin
    scalar r;
    if a < 0 then a := -a;
    if b < 0 then b := -b;
    while b neq 0 do
    begin
        r := a mod b;
        a := b;
        b := r;
    end;
    return a;
end;

procedure CoeffgcdHB(poly,mvar);
begin
    scalar res,ans;
    res := coeff(poly, mvar);
    %write res;
    ans := lcof(poly, mvar);
    foreach x in res do
```

```
        ans := HeuclidBasic(x,ans);
    return ans;
end;

procedure PrimipolyHB(x, mvar);
begin
    scalar temp;
    temp:=CoeffgcdHB(x, mvar);
    %write temp;
    if temp neq 0 then
    x:=x/temp;
    return x;
end;




procedure HEuclidBasic(a,b);
begin
    scalar r,l,xx,mvar, temp, coe1, coe2;
    if b=0 then return a;
    if a=0 then return b;
    if numberp a and numberp b then return mygcd(a,b);

    if mainvar(b) = 0 then mvar := mainvar(a) else mvar:=mainvar(b);


    if deg(a, mvar) < deg(b, mvar) then
    begin
        temp := a; a := b; b := temp
    end;

    coe1 := CoeffgcdHB(a,mvar);
    coe2 := CoeffgcdHB(b,mvar);
    a := a/coe1;
    b := b/coe2;
    r:=second(pseudo_divide(a,b,mvar));

    l:=list(lcof(a,mvar),lcof(b,mvar),lcof(r,mvar));

    %write r,l;
    while r neq 0 do
    begin
        %write l;
```

```
        a:=b;
        b:=r;
        %if mainvar(b) = 0 then mvar := mainvar(a) else mvar:=mainvar(b);
        r :=second(pseudo_divide(a,b,mvar));
        foreach xx in l do
        begin
            if xx neq 0 and abs(xx) neq 1 and r neq 0 and remainder(r,xx,mvar) eq 0 then
            begin
                while remainder(r,xx,mvar) eq 0
                do
                begin
                    %write second(divide(r,xx,mvar));
                    %write r;
                    %write xx;
                    r := r / xx;
                end;
            end;
        end;
        %write r;
        %r := PrimipolyHB(r,mvar);
        l := append(l,{lcof(r,mvar)});
    end;
    return primipolyHB(b,mvar)*HEuclidBasic(coe1, coe2);
end;
```

```
procedure mygcd(a,b);
begin
    scalar r;
    if a < 0 then a := -a;
    if b < 0 then b := -b;
    while b neq 0 do
    begin
        r := a mod b;
        a := b;
        b := r;
    end;
    return a;
end;
```

```
procedure CoeffgcdJH(poly,mvar);
begin
	scalar res,ans;
	res := coeff(poly, mvar);
	%write res;
	ans := lcof(poly, mvar);
	foreach x in res do
		ans := JHDgcd(x,ans);
	return ans;
end;

procedure PrimipolyJH(x, mvar);
begin
	scalar temp;
	temp:=CoeffgcdJH(x, mvar);
	%write temp;
	if temp neq 0 then
	x:=x/temp;
	return x;
end;

procedure process(lis, ele);
begin
	scalar a,iter,queue,temp,l,pos;
	if ele eq 0 then return lis;
	%write 127;
	foreach iter in lis do
	begin
		%write "iter = ",iter," ele = ",ele;
		while abs(iter) neq 1 and iter neq 0 and (remainder(ele,abs(iter)) eq 0)do
			ele := ele / iter;
	end;

	%write 128;
	lis := append(lis,{ele});
	queue := {ele,length(lis)};

	%write 129;
	while length(queue) neq 0 do
	begin
		%write queue;
		a := first(queue);
		queue := rest(queue);
```

```
        pos := first(queue);
        queue := rest(queue);
        %write a;
        if a neq 0 and abs(a) neq 1 then
        begin
            for iter := 1 step 1 until length(lis) do
            begin
                if (remainder(part(lis,iter),a) eq 0) and (iter neq pos) then
                begin
                    %while ((part(lis,iter) mod abs(a)) eq 0) do
                    while (remainder(part(lis,iter),a) eq 0) do
                    begin
                        lis:=(part(lis, iter) := part(lis, iter) / a);
    %                   write "127 ", (part(lis,iter));
                    end;
                    queue := append(queue,{part(lis,iter),iter});
                end;
            end;
        end;
    end;
    for iter := 0 step 1 until length(lis) do
    begin
    %   write(lis);
    %   write("jly = ",rest(lis));
        if first(lis) eq 0 or first(lis) eq 1 then
            lis := rest(lis)
        else
            lis := append(rest(lis),{first(lis)});
    end;
    return lis;

end;


procedure JHDgcd(a,b);
begin
    scalar r,l,xx,mvar, temp, coe1, coe2;
    if b=0 then return a;
    if a=0 then return b;
    if numberp a and numberp b then return mygcd(a,b);

    if mainvar(b) = 0 then mvar := mainvar(a) else mvar:=mainvar(b);
```

```
if deg(a, mvar) < deg(b, mvar) then
begin
        temp := a; a := b; b := temp
end;

coe1 := CoeffgcdJH(a,mvar);
coe2 := CoeffgcdJH(b,mvar);
a := a/coe1;
b := b/coe2;
%write "a = ",a," b = ",b;
r:=second(pseudo_divide(a,b,mvar));
%       write "r = ",r;
%write "a = ",a," b = ",b," r = ",r;
l:=list(lcof(a,mvar));
l:=process(l,lcof(b,mvar));
l:=process(l,lcof(r,mvar));

%write r,l;
while r neq 0 do
begin
        %write l;
        a:=b;
        b:=r;
        r :=second(pseudo_divide(a,b,mvar));
%       write "a = ",a," b = ",b," r = ",r;
        foreach xx in l do
        begin
                if xx neq 0 and abs(xx) neq 1 and r neq 0 and pseudo_divide(r,xx,mvar) eq 0 then
                begin
                        while remainder(r,xx,mvar) eq 0
                        do
                        begin
                                %write second(divide(r,xx,mvar));
%                               write r;
%                               write xx;
                                r := r / xx;
                        end;
                end;
        end;
        %write r;
        %r := PrimipolyJH(r,mvar);
        %l := append(l,{lcof(r,mvar)});
        l := process(l, lcof(r,mvar));
end;
```

```
        b := PrimipolyJH(b,mvar);
        %write "coe1 = ",coe1, " coe2 = ",coe2;
        return b*JHDgcd(coe1, coe2);
end;

procedure mygcd(a,b);
begin
        scalar r;
        if a < 0 then a := -a;
        if b < 0 then b := -b;
        while b neq 0 do
        begin
                r := a mod b;
                a := b;
                b := r;
        end;
        return a;
end;

procedure CoeffgcdPR(poly,mvar);
begin
        scalar res,ans;
        res := coeff(poly, mvar);
        %write res;
        if length(res) eq 1 then return first(res);
        ans := lcof(poly, mvar);
        foreach x in res do
                ans := Primgcd(x,ans);
        return ans;
end;

procedure PrimipolyPR(x, mvar);
begin
        scalar temp;
        temp:=CoeffgcdPR(x, mvar);
        %write temp;
        if temp neq 0 then
        x:=x/temp;
        return x;
end;

procedure Primgcd(a,b);
begin
        scalar coe1,coe2,coef,temp, mvar;
```

```
	%write a;
	%write b;
	if b = 0 then return a;
	if a = 0 then return b;
	if numberp b and numberp a then return mygcd(a,b);

	if mainvar(b) = 0 then mvar := mainvar(a) else mvar:=mainvar(b);
	%write mvar;
	if deg(a, mvar) < deg(b, mvar) then
	<<temp:=a;a:=b;b:=temp>>;
	%write 123;
	coe1 := CoeffgcdPR(a,mvar);
	coe2 := CoeffgcdPR(b,mvar);
	%write 124;
	coef := Primgcd(coe1, coe2);
	%write 125;
	if coe1 neq 0 then a:=a/coe1;
	if coe2 neq 0 then b:=b/coe2;


	r:=PrimipolyPR(second((pseudo_divide(a,b,mvar))),mvar);
	while r neq 0 do
	begin
		a:=b;
		b:=r;
		r:=PrimipolyPR(second((pseudo_divide(a,b,mvar))),mvar);
		%write r;
	end;
	return coef * b;
end;

procedure mygcd(a,b);
begin
	scalar r;
	if a < 0 then a := -a;
	if b < 0 then b := -b;
	while b neq 0 do
	begin
		r := a mod b;
		a := b;
		b := r;
	end;
	return a;
end;
```

```
procedure CoeffgcdSR(poly,mvar);
begin
	scalar res,ans;
	res := coeff(poly, mvar);
	%write res;
	if length(res) eq 1 then return first(res);
	ans := lcof(poly, mvar);
	foreach x in res do
		ans := SREuclid(x,ans);
	return ans;
end;

procedure PrimipolySR(x, mvar);
begin
	scalar temp;
	if mvar eq 0 then return x;

	temp:=CoeffgcdSR(x, mvar);
	%write temp;
	if temp neq 0 then
	x:=x/temp;
	return x;
end;


procedure SREuclid(f,g);
begin
	scalar beta0,
	a0, a1,a2,coe1,coe2,
	del0, del1, del2,
	psi0, psi1, psi2,mvar;
	if g = 0 then return f;
	if f = 0 then return g;
	if numberp g and numberp f then return mygcd(g,f);

	%array mybeta(5), a(5),
	%mydel(5), mypsi(5);
	if mainvar(g) = 0 then mvar := mainvar(f) else mvar:=mainvar(g);

	if deg(f,mvar) < deg(g,mvar) then
	begin
		a0 := PrimipolySR(g,mvar);
		a1 := PrimipolySR(f,mvar);
```

```
        coe1 := g/a0;
        coe2 := f/a1;
    %   write coe1;
    %   write coe2;
    end
    else
    begin
        a0 := PrimipolySR(f,mvar);
        a1 := PrimipolySR(g,mvar);
        coe2 := f/a0;
        coe1 := g/a1;
    %   write coe1;
    %   write coe2;
    end;

    del0 := deg(a0,mvar) - deg(a1,mvar);
    beta0 := (-1)^(del0+1);
    psi0:=-1;
    while a1 neq 0 do
    <<
        %write "a0=",a0;
        %write "a1=",a1;
        %write "mvar=",mvar;
        %write pseudo_remainder(a0,a1,mvar);
        a2 := first divide(pseudo_remainder(a0, a1, mvar), beta0, mvar);;
        %write a2," ",beta0;
        %write 127;
        del1 := deg(a1,mvar) - deg(a2,mvar);
        psi1 := (-lcof(a1,mvar))^del0 * psi0^(1-del0);
        %write 128;
        beta0 := -lcof(a1,mvar)*psi1^del1;
        %write 129;
        a0 := a1;
        a1 := a2;
        del0 := del1;
        psi0 := psi1;
        >>;
    %write 130;
    %write a0;
    %write mvar;
    %write coe1;
%   write coe2;
    if mvar neq 0 then
    temp := PrimipolySR(a0,mvar)
```

```
    else
    temp:=1;

    %write 127;
    return temp*SREuclid(coe1,coe2);
end;
```

[i] Wikipedia https://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor