Approximation de la fonction racine carrée

2) La méthode de dichotomie

1) Au départ, l'intervalle [a0,b0]=[a,b] est de longueur b-a.
A chaque étape, la longueur de l'intervalle [an,bn] est divisée par 2.
D'où :

```
> b[n]-a[n]=(b-a)/2^n;
```

$$b_n - a_n = \frac{b-a}{2^n}$$

2) La fonction x->x^2-A s'annule en plus ou moins racine carrée de A.

```
> f:=x->x^2-A;
```

$$f := x \to x^2 - A$$

3)

```
> Dichotomie:=proc(A,epsilon)
                local f,an,bn,m;
  f:=x->x^2-A;
  an:=0.;
  bn:=evalf((A+1)/2);
  while bn-an >epsilon do
  m:=(an+bn)/2;
  if f(m)<= 0 then # On exploite les variations de f pour
  simplifier le test.
     an:=m
  else
     bn:=m
  fi;
  od;
  [an,bn];
  end;
```

Dichotomie := **proc**(*A*, ε)
local *f*, *an*, *bn*, *m*;
f := *x* → *x*^2 − *A*;
an := 0.;
bn := evalf(1 / 2∗*A* + 1 / 2);
while ε < *bn* − *an* **do**
m := 1 / 2∗*an* + 1 / 2∗*bn*; **if** *f*(*m*) ≤ 0 **then** *an* := *m* **else** *bn* := *m* **end if**
end do;
[*an*, *bn*]
end proc

4)

```
> Dichotomie(3,10^(-3));
  Dichotomie(3,10^(-4));
```

```
Dichotomie(3,10^(-5));
Dichotomie(3,10^(-6));
Dichotomie(3,10^(-7));
Dichotomie(3,10^(-8));
```

$[1.731445312, 1.732421875]$

$[1.731994629, 1.732055664]$

$[1.732048034, 1.732055664]$

$[1.732049942, 1.732050896]$

$[1.732050777, 1.732050836]$

$[1.732050806, 1.732050814]$

5)

```
> Dichotomie:=proc(A,epsilon)
                local f,an,bn,m,N;
  f:=x->x^2-A;
  an:=0.;
  bn:=evalf((A+1)/2);
  N:=4;  # 2 affectations + 2 opérations
  while bn-an >epsilon do
     m:=(an+bn)/2;       # 1 affectation + 2 opérations
      if f(m)<= 0 then   # 1 test + 2 opérations + 1 affectations

         an:=m
     else
        bn:=m
     fi;
     N:=N+7;
  od;
  [[an,bn],N];
  end;
```

Dichotomie := **proc**(*A*, ε)

local *f*, *an*, *bn*, *m*, *N*;

f := *x* → *x*^2 − *A*;

an := 0.;

bn := evalf(1 / 2∗*A* + 1 / 2);

N := 4;

while ε < *bn* − *an* **do**

m := 1 / 2∗*an* + 1 / 2∗*bn*; **if** *f*(*m*) ≤ 0 **then** *an* := *m* **else** *bn* := *m* **end if**; *N* := *N* + 7

end do;

[[*an*, *bn*], *N*]

end proc

6)

```
> Dichotomie(3,10^(-3));
  Dichotomie(3,10^(-4));
```

```
Dichotomie(3,10^(-5));
Dichotomie(3,10^(-6));
Dichotomie(3,10^(-7));
Dichotomie(3,10^(-8));
```

[[1.731445312, 1.732421875], 81]

[[1.731994629, 1.732055664], 109]

[[1.732048034, 1.732055664], 130]

[[1.732049942, 1.732050896], 151]

[[1.732050777, 1.732050836], 179]

[[1.732050806, 1.732050814], 200]

3) La méthode de Théon de Smyrne

1)

```
> Theon:=proc(A,epsilon)
        local rp,ri,f;
  # rp contiendra les termes de rangs pairs de la suite
  # ri contiendra les termes de rangs impairs de la suite
  f:=x->evalf((x+A)/(x+1));
  rp:=1.;
  ri:=f(rp);
  while abs(rp-ri)>epsilon do
  # Au départ, rp contient r_(2n) et ri contient r_(2n+1)
     rp:=f(ri); # rp contient maintenant r_(2n+2)
     ri:=f(rp); # ri contient maintenant r_(2n+3)
  od;
  [rp,ri];
  end;
```

Theon := **proc**(*A*, ε)

local *rp*, *ri*, *f*;

f := *x* → evalf((*x* + *A*) / (*x* + 1));

rp := 1.;

ri := *f*(*rp*);

while ε < abs(*rp* − *ri*) **do** *rp* := *f*(*ri*); *ri* := *f*(*rp*) **end do**;

[*rp*, *ri*]

end proc

2)

```
> Theon(3,10^(-3));
  Theon(3,10^(-4));
  Theon(3,10^(-5));
  Theon(3,10^(-6));
  Theon(3,10^(-7));
  Theon(3,10^(-8));
```

[1.731707317, 1.732142857]

[1.732026144, 1.732057416]

[1.732049037, 1.732051282]

[1.732050680, 1.732050842]

[1.732050798, 1.732050810]

[1.732050807, 1.732050808]

3)

```
> Theon:=proc(A,epsilon)
        local rp,ri,f,N;
  if A=0 then [[0.,0.],0]
  else
  f:=x->evalf((x+A)/(x+1)); # Une évaluation de f demande 3
  opérations
  rp:=1.;
  ri:=f(rp);
  N:=5; # 2 affectations et 3 opérations
  while abs(rp-ri)>epsilon do
     rp:=f(ri);
     ri:=f(rp);
     N:=N+10; # 2 affectations, 6 opérations et 2 opérations pour
  le test (on ne compte pas le calcul de la valeur absolue car un
  test initial permettrait de s'en passer)
  od;
  [[rp,ri],N];
  fi;
  end;
```

Theon := **proc**(*A*, ε)

local *rp*, *ri*, *f*, *N*;

if *A* = 0 **then** [[0., 0.], 0]

else

f := *x* → evalf((*x* + *A*) / (*x* + 1));

rp := 1.;

ri := *f*(*rp*);

N := 5;

while ε < abs(*rp* − *ri*) **do** *rp* := *f*(*ri*); *ri* := *f*(*rp*); *N* := *N* + 10 **end do**;

[[*rp*, *ri*], *N*]

end if

end proc

4)

```
> Theon(3,10^(-3));
  Theon(3,10^(-4));
  Theon(3,10^(-5));
  Theon(3,10^(-6));
```

```
Theon(3,10^(-7));
Theon(3,10^(-8));
```

[[1.73170731707317, 1.73214285714286], 35]

[[1.73202614379085, 1.73205741626794], 45]

[[1.73204903677758, 1.73205128205128], 55]

[[1.73205068043172, 1.73205084163518], 65]

[[1.73205079844084, 1.73205081001473], 75]

[[1.73205080691351, 1.73205080774448], 85]

4) La méthode de Héron d'Alexandrie

1)

```
> Heron:=proc(A,epsilon)
        local u,v;
  u:=evalf((A+1)/2);
  v:=evalf(A/u);
  while u-v>epsilon do
     u:=evalf((u+v)/2); # On remarque que u_(n+1)=(u_n+v_n)/2 où
  v_n=A/u_n
     v:=evalf(A/u);
  od;
  [u,v];
  end;
```

Heron := **proc**(*A*, ε)

local *u*, *v*;

u := evalf(1 / 2∗*A* + 1 / 2);

v := evalf(*A* / *u*);

while ε < *u* − *v* **do** *u* := evalf(1 / 2∗*u* + 1 / 2∗*v*); *v* := evalf(*A* / *u*) **end do**;

[*u*, *v*]

end proc

2)

```
> Heron(3,10^(-3));
  Heron(3,10^(-4));
  Heron(3,10^(-5));
  Heron(3,10^(-6));
  Heron(3,10^(-7));
  Heron(3,10^(-8));
```

[1.732142857, 1.731958763]

[1.732050810, 1.732050805]

[1.732050810, 1.732050805]

[1.732050810, 1.732050805]

[1.732050810, 1.732050805]

[1.732050810, 1.732050805]

3)

```
> Heron:=proc(A,epsilon)
        local u,v,N;
  u:=evalf((A+1)/2);
  v:=evalf(A/u);
  N:=5; # 3 opérations et 2 affectations
  while u-v>epsilon do
     u:=evalf((u+v)/2); # On remarque que u_(n+1)=(u_n+v_n)/2 où
  v_n=A/n+u_n
     v:=evalf(A/u);
     N:=N+7; # 4 opérations, 2 affectations et 1 test
  od;
  [[u,v],N];
  end;
```

Heron := **proc**(*A*, ε)

local *u*, *v*, *N*;

u := evalf(1 / 2∗*A* + 1 / 2);

v := evalf(*A* / *u*);

N := 5;

while ε < *u* − *v* **do** *u* := evalf(1 / 2∗*u* + 1 / 2∗*v*); *v* := evalf(*A* / *u*); *N* := *N* + 7 **end do**;

[[*u*, *v*], *N*]

end proc

4)

```
> Heron(3,10^(-3));
  Heron(3,10^(-4));
  Heron(3,10^(-5));
  Heron(3,10^(-6));
  Heron(3,10^(-7));
  Heron(3,10^(-8));
```

[[1.732142857, 1.731958763], 19]

[[1.732050810, 1.732050805], 26]

[[1.732050810, 1.732050805], 26]

[[1.732050810, 1.732050805], 26]

[[1.732050810, 1.732050805], 26]

[[1.732050810, 1.732050805], 26]

5) La méthode de Héron d'Alexandrie

1)

```
> gD:=A->Dichotomie(A,10^(-8))[2];
  gT:=A->Theon(A,10^(-8))[2];
  gH:=A->Heron(A,10^(-8))[2];
  Digits:=15; # Nous augmentons le nombre de chiffres
  significatifs pour que la précision soit ateignable.
```

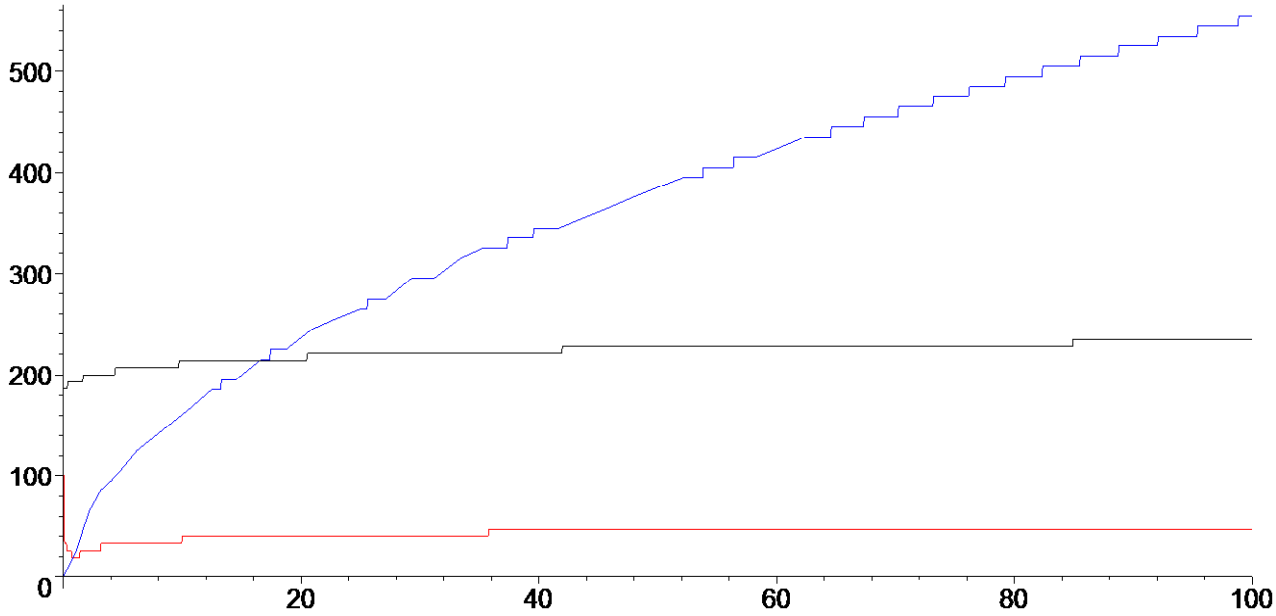
$$gD := A \to \text{Dichotomie}\left(A, \frac{1}{100000000}\right)_2$$

$$gT := A \to \text{Theon}\left(A, \frac{1}{100000000}\right)_2$$

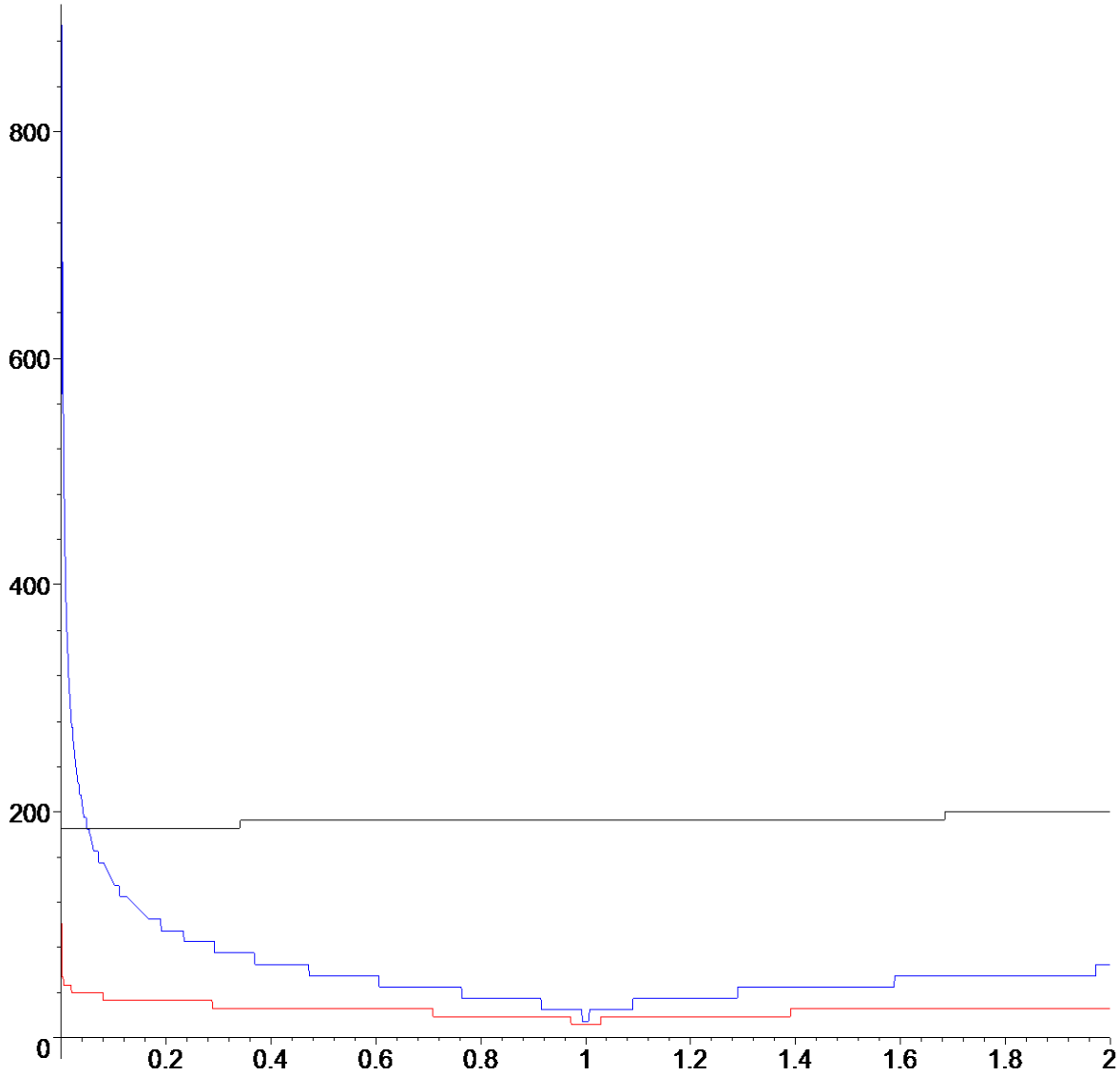
$$gH := A \to \text{Heron}\left(A, \frac{1}{100000000}\right)_2$$

$$Digits := 15$$

```
> plot([gD,gT,gH],0..100,color=[black,blue,red]);
```




```
> plot([gD,gT,gH],0..2,color=[black,blue,red]);
```



2) On remarque ainsi que la méthode de Théon est la plus mauvaise pour les petites et les grandes valeurs. Elle est cependant meilleure que la méthode de Dichotomie pour des valeurs comprise entre 0,2 et 15 environ.
La méthode de Héron semble être la meilleure des trois méthodes.