Using the HP48G solver and the LongFloat Library for extended precision equation solutions (HP48G)

Ph.J.Roussel, *PCX* Belgium (1)

Introduction

The intention of this paper is twofold:

1. Respond to a challenge presented by Benny Vanrutten at a ***PCX*** meeting back in 1998.

2. Show how the standard HP48G equation solver can be used to find extended precision solutions to equations defined with LongFloat library commands.

The challenge: Speedy Gonzales

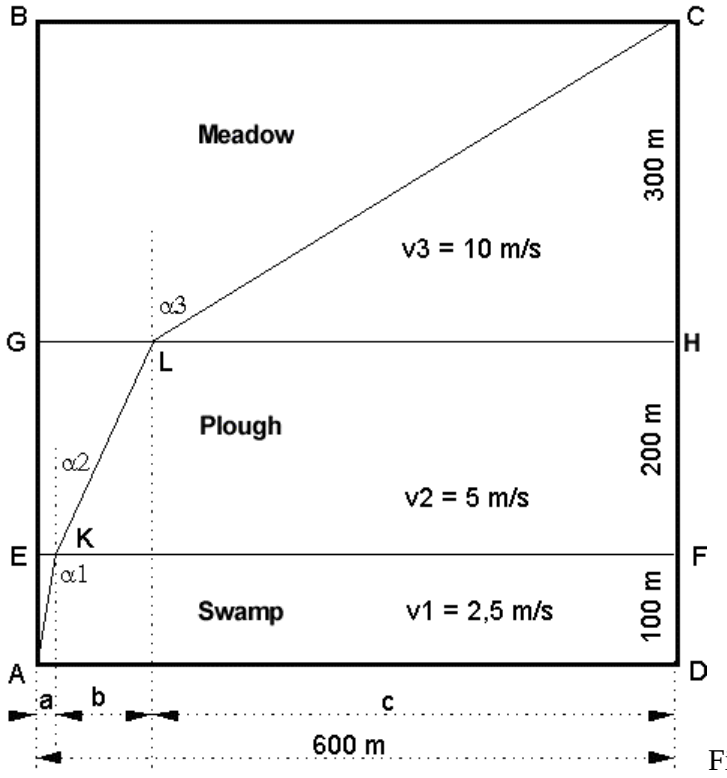

Fig. 1

Fig. 1 above represents a square piece of land with sides 600 m long.

1. The land consists of three fields: a meadow, a plough, and a swamp.
2. The rectangle AEFD represents the swamp. The side DF is 100 m long.
3. The rectangle EGHF represents the plough. The side FH is 200 m long.
4. The rectangle GBCH represents the meadow. The side HC is 300 m long.
5. A farmer wants to get from A to C as fast as possible, given the swamp only allows a v1=2.5 m/s crossing speed, while the plough can be crossed at v2=5 m/s, and the meadow at v3=10 m/s. By the way, the latter rather unrealistic speed prompted me to name the puzzle "Speedy Gonzales" ☺

The challenge: what's the shortest possible time for the farmer to get from A to C, and on top of that, compute it to 48 digits of accuracy?

If I'm well informed, the challenge was already published in the Leisure Lines column of Personal Computer World in 1981. Stephan Grant from Horsham, England won the contest then.

The optimal path equation

It should be clear that running from A to C diagonally is not the fastest way, due to the different field crossing speeds. Obviously, the optimal trajectory consists of straight line segments, one for each field type. People that recognise a figure from an optics course in fig. 1 can skip an elaborate theoretical part of the solution process: indeed, it can be shown that the relationships between the incidence angles α_1, α_2 and α_3 for the optimal path obey the equivalent of Snell's law:

$$\frac{\mathrm{Sin}(\alpha_1)}{v_1} = \frac{\mathrm{Sin}(\alpha_2)}{v_2} = \frac{\mathrm{Sin}(\alpha_3)}{v_3} \quad (1)$$

Snell's law gives the relationships between the incidence angles of a light beam travelling through a sequence of planparallel media with different refractive indices, or light propagation speeds. As yet another challenge (for potential disbelievers), I deduced this law symbolically on an HP48G equipped with the ALG48 library of Claude-Nicholas Fiechter and Mika Heiskanen. The result was published earlier in PCXJV13N2P4. In the analogous optics case, the external incidence angle is commonly given, while for Speedy Gonzales, it is the field width AD:

$$DF \cdot \mathrm{Tan}(\alpha_1) + FH \cdot \mathrm{Tan}(\alpha_2) + HC \cdot \mathrm{Tan}(\alpha_3) = AD \quad (2)$$

Two of the three incidence angles can be eliminated through substitution of Eq.(1) into Eq.(2).

Thus, the problem is reduced to the solution of a single non-linear equation in one of the incidence angles, e.g. α_3. As trigonometric functions are time consuming, and bearing in mind that this is even more so when computing them with extended precision, expressing the equation in terms of a single trigonometric transform variable is quite advantageous. Thus, the optimal path equation to be solved as a function of $sa3 = \mathrm{Sin}(a_3)$ becomes (presented in HP48 EquationWriter format):

$$\frac{DF\cdot\left(\frac{v1}{v3}\cdot s\alpha 3\right)}{\sqrt{1-\left(\frac{v1}{v3}\cdot s\alpha 3\right)^2}}+\frac{FH\cdot\left(\frac{v2}{v3}\cdot s\alpha 3\right)}{\sqrt{1-\left(\frac{v2}{v3}\cdot s\alpha 3\right)^2}}+\frac{HC\cdot s\alpha 3}{\sqrt{1-s\alpha 3^2}}-AD=0 \qquad (3)$$

The next equation computes a travelling time for a specific trajectory even for non-optimal paths so that it can be used to verify that the solution of Eq.3 in 'sα3' is minimal. It looks a bit awkward at first, but to make the validity range of the equation as wide as possible, the trajectory is defined as follows: for a given value of α_3, start crossing the first field from A to K at an incidence angle α_1 computed as a function of α_3 using Snell's law (Eq.1). Then continue from K to L at incidence angle α_2 computed as a function of α_3 using the same law. Then complete the trajectory from L to C (which will only be at incidence angle α_3 for the optimal α_3 value). The travelling time equation (4) as a function of 'sα3' then becomes:

$$\frac{DF}{v1\sqrt{1-\left(\frac{v1}{v3}\cdot s\alpha 3\right)^2}}+\frac{FH}{v2\sqrt{1-\left(\frac{v2}{v3}\cdot s\alpha 3\right)^2}}+\frac{\sqrt{\left(AD-\frac{DF\cdot\left(\frac{v1}{v3}\cdot s\alpha 3\right)}{\sqrt{1-\left(\frac{v1}{v3}\cdot s\alpha 3\right)^2}}-\frac{FH\cdot\left(\frac{v2}{v3}\cdot s\alpha 3\right)}{\sqrt{1-\left(\frac{v2}{v3}\cdot s\alpha 3\right)^2}}\right)^2+HC^2}}{v3}$$

Now, this time can be plotted for α_3 from 45° ($sa3=\sqrt{2}/2$) to 90° ($s\alpha 3=1$):

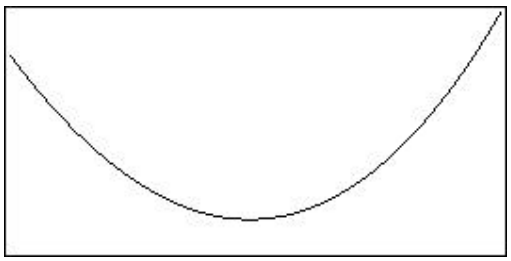

Fig.2 Plot of travelling time $t(sa3)$ computed with standard precision.

The plot indeed shows a minimum near $s\alpha 3 \approx 0.85$

The HP48 LongFloat library

Claude-Nicholas Fiechter and Mika Heiskanen also designed a supplement to the ALG48 library that offers extended precision floating point arithmetic. Both libraries can be downloaded from the Internet. The ***HPCC*** and ***PCX*** websites also offer pointers to that website. The LongFloat library only takes 6 kB of RAM, but it needs some routines from ALG48, which takes 49 kB (version 4.2). The extended precision numbers are stored as string objects, the extended precision arithmetic operations are available as a separate command set. The logic behind the command names: take the standard precision command preceded by an F. The user can adjust the precision required through an extra reserved 'DIGITS' variable stored in the HOME directory. Some sample computations:

HOME 48 'DIGITS' STO "-1" FACOS results in

"3.14159265358979323846264338327950288419716939938"

"10000" FEXP DUP FLN puts the extended precision numbers

"8.80681822566292158726149600764456100352000408944E4342"

"10000" on the stack.

Combining LongFloat equations with the standard HP48 solver

The HP48 solver is of course only designed to operate with standard BCD floating point real numbers, both for the variables and the output of the algebraic or the program object defining the equation. Any extended precision computation result in string format has to be converted into a BCD real number using the OBJ¨ or the STR¨ command, causing the loss of the tail digits:

"0.850126480556973401241036 2" STR¨

results in the BCD real number .850126480557

However, in the context of equation solving, we can use this to our advantage. If an equation is defined as a function in the reserved EQ variable, the right hand side of the equation defaults to zero. In many cases, the leading digits of the function output start to vanish in the vicinity of its root due to subtractive cancellation. For very sensitive functions, the output of a standard precision function may return zero before a sufficient accuracy is achieved for the standard precision variable. For less sensitive functions, it is often impossible to find a standard precision value for the MCALC variable returning a zero for the function value. Such limitations have to be intercepted in numerical solver algorithms, so that extra stopping criteria like the one leading to the Sign Reversal message have to be foreseen.

For a function definition limited by subtractive cancellation like Eq.3 above, more and more of the leading digits of the terms in the algebraic sum computed with extended precision cancel out as the solver approaches the function root, while the OBJ¨ command performing the object type conversion required still results in a full 13 digit BCD real instead of a zero.

In most cases the input variables fed to the extended precision function also need to be stored with extended accuracy. This can simply be achieved by composing them from standard precision terms with non-overlapping mantissa ranges, e.g.

```
'.850126480557-2.65987589638E-14-4.01748436027E-26-2.37828007315E-38'
```

For the Speedy Gonzales problem at hand, only 'sα3' needs to be encoded as 'sα31+sα32+sα33+sα34' for the 48 digits precision required, as all of the MUSER parameters can be handled with standard precision.

Once the equation is encoded using the LongFloat library commands with the 'sα3' components as the input variables, it can be solved for each of them in sequence with the HP48 Equation Solver, starting from the standard precision term down to the highest precision term.

Utility programs

UREF$ converts a unit object input parameter into the corresponding standard SI reference numeric value, and then into a string object for further processing with LongFloat commands.

```
'UREF$'  40 bytes  Checksum: # C87Bh
« ¨ Uvar 'UVAL(UBASE(Uvar))' ¨STR
»
```

Extended precision function that maps vertical projection d to the corresponding distance travelled in a field with a given speed ratio vr for given sΘ3 value x3.

```
'v¨d'  84.5 bytes  Checksum: # F7CBh
« ¨ d x3 vr
  « d "1" x3 vr FMUL DUP FMUL FSUB FSQRT FDIV
  »
»
```

Used for the computation of terms like $\frac{DF}{\sqrt{1-\left(\frac{v1}{v3}\cdot s\alpha 3\right)^2}}$

Extended precision function that maps vertical projection `d` to the corresponding horizontal projection for a field with a given speed ratio `vr` for given `sΣ3` value `x3`.

```
'v¨h'  92.5 bytes  Checksum: # 5D4Ch
« ¨ d x3 vr
  « x3 vr FMUL "1" OVER DUP FMUL FSUB FSQRT FDIV d FMUL
  »
»
```

Used for the computation of terms like $\frac{DF\cdot\left(\frac{v1}{v2}\right)\cdot s\alpha 3}{\sqrt{1-\left(\frac{v1}{v3}\cdot s\alpha 3\right)^2}}$

With the above utilities, Eq.3 can be programmed as

```
'FsΣ3$'  323 bytes  Checksum: # 1E0Dh
« ¨ sΣ3
  « 'v¨h(UREF$(DF),sΣ3,UREF$(v1/v3))' EVAL
'v¨h(UREF$(FH),sΣ3,UREF$(v2/v3))' EVAL FADD
'v¨h(UREF$(HC),sΣ3,UREF$(1))' EVAL FADD 'UREF$(AD)' EVAL FSUB OBJ¨
  »
»
```

Travelling time equation 4 can be programmed in a similar way:

```
't$.EQ'  593 bytes  Checksum: # 8BB0h
« ¨ sΣ3
  « 'v¨d(UREF$(DF),sΣ3,UREF$(v1/v3))' EVAL 'UREF$(v1)' EVAL FDIV
'v¨d(UREF$(FH),sΣ3,UREF$(v2/v3))' EVAL 'UREF$(v2)' EVAL FDIV FADD
'UREF$(AD)' EVAL 'v¨h(UREF$(DF),sΣ3,UREF$(v1/v3))' EVAL FSUB
'v¨h(UREF$(FH),sΣ3,UREF$(v2/v3))' EVAL FSUB DUP FMUL 'UREF$(HC)'
EVAL DUP FMUL FADD FSQRT 'UREF$(v3)' EVAL FDIV FADD
  »
»
```

The function of the 4 `sΣ3` components to be stored in `'EQ'` becomes:

```
'F4'  239 bytes  Checksum: # EB06h
« ¨ sΣ31 sΣ32 sΣ33 sΣ34
  « 'UREF$(sΣ31)' EVAL 'UREF$(sΣ32)' EVAL 'UREF$(sΣ33)' EVAL
'UREF$(sΣ34)' EVAL
  » FADD FADD FADD ¨ sΣ3$ 'FsΣ3$(sΣ3$)'
»
```

The `F4` equation can be solved for `sΣ3`.with the input parameters defined using

```
'IniPars'  246.5 bytes  Checksum: # D250h
« { '600_m' '100_m' '200_m' '300_m' '2.5_m/s' '5_m/s' '10_m/s' }
{ AD DF FH HC v1 v2 v3 } STO »
```

The Multiple Equation Solver approach

The Multiple Equations Solver (MES) can also be used to automate the solution process. With the following set of equations stored in 'EQ':

```
'sŒ3.EQ'  162 bytes  Checksum: # A0CBh
{ 'F4(sŒ31,sŒ32,sŒ33,sŒ34)' 'F3(sŒ31,sŒ32,sŒ33)'
'F2(sŒ31,sŒ32)' 'F1(sŒ31)' }
```

the MES automatically finds the single precision equation F1 and solves it for sŒ31, then finds F2 and solves that for sŒ32, and so on for all sŒ3 components.

F1, F2 and F3 still have to be defined:

```
'F1'  92 bytes  Checksum: # 8904h
« ¨ sŒ31 'UREF$(sŒ31)' ¨ sŒ3$ 'FsŒ3$(sŒ3$)'
»
'F2'  146 bytes  Checksum: # A1Bh
« ¨ sŒ31 sŒ32
  « 'UREF$(sŒ31)' EVAL 'UREF$(sŒ32)' EVAL
  » FADD ¨ sŒ3$ 'FsŒ3$(sŒ3$)'
»
'F3'  192.5 bytes  Checksum: # 66E0h
« ¨ sŒ31 sŒ32 sŒ33
  « 'UREF$(sŒ31)' EVAL 'UREF$(sŒ32)' EVAL 'UREF$(sŒ33)' EVAL
  » FADD FADD ¨ sŒ3$ 'FsŒ3$(sŒ3$)'
»
```

The LNGSLV program below initialises the MES for solution of Eq.3. To assure convergence of the Solver, a proper starting value for sŒ3 has to be stored in sŒ31. The 2/ƒ5 value corresponding with the following strategy works well: first cross the swamp and the plough at zero incidence angles, then cross the meadow diagonally. The travelling time can be computed with the following Ft function

```
'Ft'  237 bytes  Checksum: # 70FFh
« ¨ sŒ31 sŒ32 sŒ33 sŒ34
  « 'UREF$(sŒ31)' EVAL 'UREF$(sŒ32)' EVAL FADD 'UREF$(sŒ33)' EVAL
FADD 'UREF$(sŒ34)' EVAL FADD ¨ x3$ 't$.EQ(x3$)'
  »
»
'LNGSLV'  305 bytes  Checksum: # 67DAh
« PATH HOME 48 'DIGITS' STO EVAL STD 'sŒ3.EQ' STEQ MINIT IniPars
"Precision Gonzales" { sŒ31 sŒ32 sŒ33 sŒ34 } MITM '2/ƒ5' EVAL 1
¨LIST { 0 0 0 } + { sŒ31 sŒ32 sŒ33 sŒ34 } STO MSOLVR HALT
'Ft(sŒ31,sŒ32,sŒ33,sŒ34)' EVAL DUP 'tm' STO "tm" ¨TAG
»
```

The program HALTs with the MSOLVR menu ready.

The leftshift `All` softkey then starts the solution process for the 4 `sC3` components. As specified in the manual, the root approximations are shown upon hitting the `ENTER` key during the solution of any of the 4 equations. When doing so, some rather strange intermediate displays appear when solving for the high precision components with zero starting values. The Solver apparently needs some special precautions to recover from the fact that the function is rather insensitive to them, but nevertheless, it finds all root components!

Finally, `CONT`inuing the `LNGSLV` program results in the minimal travelling time with 48-digit precision:

```
tm: "142.097659044197040067245550187723934324991685915"
```