Is there Honest Perplexity with your Hyperlunar Passage? If so, use Hardy Persistence!

See page 32 for program. Delmer D. Hinrichs (967)

Celestial Mechanics Simulation: "Lunar Landing from Random Orbit"

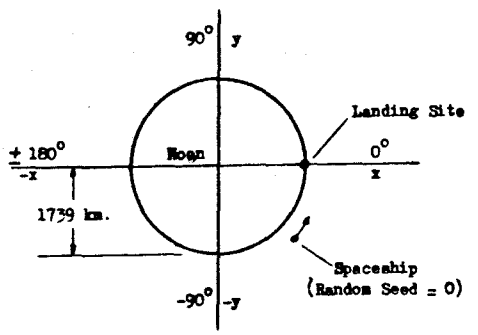
This challenging game simulates the landing of a spaceship at a predetermined target location on the moon. The initial spaceship position is randomly chosen to be in any direction from the moon, at a radius of 2,000 km. to 3,000 km. from the moon's center (lunar radius is 1,739 km.). The random elliptical orbit is always counter-clockwise. Warning: some orbits will eventually impact on the lunar surface, if not modified.

This program accurately calculates the initial orbit and landing trajectory of a spaceship. This allows the user to both observe some of the peculiarities of celestial mechanics, and to have an engrossing game program. Some realistic features of the program are:

1. Real, 2-D spaceship trajectory is calculated as a series of segments of user-specified duration. Segment duration may be any integer or decimal number of seconds. (Accuracy is better with shorter segments).
2. True, inverse-square-law gravity is calculated, acting upon the approximate segment mid-point. Gravity at the lunar surface is 1.62 m./sec.^{2}, the correct value.
3. Any integer number of segments of free fall may be automatically calculated.
4. The spaceship has a rocket engine with an exhaust velocity of 4 km./sec., simulating a hydrogen-oxygen fuel.
5. The spaceship gets lighter as fuel is used. Therefore maximum acceleration increases from about 1.3 g's to about 4 g's as fuel is used (with a 100 kg./sec. fuel usage rate).
6. Total fuel usage is limited to 20,000 kg., which, along with the dry mass of 10,000 kg., results in a 3 : 1 mass ratio.
7. Fuel usage rate may be any integer or decimal number of kg./sec. (For realism, maximum usage should be 100 kg./sec.).
8. An attempted burn greater than the remaining fuel results in a signal, and only the actual remaining fuel is burned.
9. An attempted burn when fuel is gone results in free fall to impact.
10. Impact (or landing) on the lunar surface results in a signal, and then landing conditions are automatically calculated. Distance along the lunar surface to the target site, and direction and velocity of the spaceship at the moment of impact are interpolated from the segment-end conditions immediately before and after impact.

Note that the direction of spaceship travel, and its angular position, are with respect to the fixed frame of reference of the lunar center (origin) and the target landing site (0°). See sketch.

On an HP-97, the new status will automatically print out after each move, as will the impact signal and the impact conditions.



Sketch for: "Lunar Landing from Random Orbit"

See page 36 for program.

Delmer D. Hinrichs (967)

"Celestial Mechanics Demonstration - Robot Rocket Lander"

This two-card program is intended for those "Rocket Lander" pilots that sometimes have difficulty estimating how to make a landing from orbit. With this program, all you have to do is to start it, and then let it figure out what to do.

Initially, the spaceship is in a circular orbit at a variable altitude above Mars or the Moon, and must land at a preselected target landing site. This program calculates the descent as a series of segments of varying duration. As required, the program automatically adjusts segment time up or down, decides for each segment whether it should be a free fall or a rocket burn segment, and selects the angle of thrust for rocket burns.

The descent is made in five stages: 1) Wait in circular orbit until opposite the target site (the initial positions are all close to this point to avoid long waits); 2) Make a rocket burn for a Hohmann transfer orbit (for a minimum-fuel-usage altitude change); 3) Wait in elliptical transfer orbit until time to: 4) Stop orbital motion above the target site, then: 5) Make a final, vertical descent onto target.

Realistic features of this program include: 1) True inverse-square-law gravity is calculated, acting upon each segment's midpoint; 2) Rocket engine thrust simulates the use of a hydrogen-oxygen fuel, with an exhaust velocity of 4 km./sec.; 3) The spaceship loses mass as fuel is used; with less inertia, the ship's acceleration rate increases; 4) The ship's position is updated using its average direction and velocity for the segment.

The equations used to estimate whether to free fall or to make a rocket burn for the next segment, and when to reduce segment duration, are basically quite simple:

$S_i = A/Vt$ when: S_i = Est. seg. free fall to impact

$S_s = V/\Delta V$ S_s = Est. seg. rocket burn to stop

A = Altitude

Free fall if: $S_s < 2(S_i - 1)$ V = Ship's velocity

Red. seg. if: $S_i < 1$ t = Segment duration

or if: $S_s < 1$ ΔV = Vel. change from rocket burn

These equations are first used to stop horizontal velocity over the target site, then again to stop vertical velocity at the surface. When in the final vertical descent, V must be corrected for its anticipated increase due to gravity during the next segment, and ΔV must be corrected for its decrease due to gravity.

If you want to try these descents manually (they can be improved upon), initialize for the desired orbit, then load whichever of my previous programs, "Random Lunar Lander" or "Mars Lander", is appropriate and go. Do not reinitialize! Note that the sign of the gravity constant of "Random Lunar Lander" must be changed before doing a manual descent with this initialization (i.e., "RCL B, CHS, STO B"). Also note that the initial fuel supply will be less than for the normal initialization, so be careful!

See page 35 for program. Delmer D. Hinrichs (967)

"Flip", a Learning Game

This game appears very simple, but is actually rather difficult. The HP-67 chooses heads or tails (1 or 0), and the user then tries to guess the HP-67's choice. At first, the odds are even (50-50) on each guess. However, the HP-67 gradually determines the patterns in the user's guesses, and then chooses the opposite of what the user is predicted to choose. Actually, the HP-67 does not strictly maximize its chances on each move, but instead includes some randomness, to make it harder for the user to predict the HP-67's choice. This game thus simulates the type of two-person game in which each tries to outguess the other's behavior and to stay one step ahead.

This learning program differs from others such as "Hexapawn", "Mini-Checkers", "Cybernim", etc. in that:

1. "Learning" does not completely exclude the possibility of a given move, but merely shifts the probabilities.
2. The program learns from every trial, not only from losing trials (i.e., there is both positive and negative learning).
3. Learning is not absolutely fixed; what has been learned may be unlearned later, if conditions change.

This program keeps track of its last two choices, and of the user's last two guesses in response. As these four bits of information define 16 different cases, the program also keeps an array of 16 different probabilities (in R4 through R19). Each of these probabilities is initially set to 0.5, representing even odds. After each user guess, the corresponding probability is adjusted by 20% (i.e., from 0.5 to 0.4, or from 0.6 to 0.68). The probability is adjusted upwards or downwards depending upon whether or not the user's guess was correct. Gradually, the array of 16 probabilities becomes a profile of the user's probable responses, and is used by the HP-67 to choose the opposite. Any probabilities very far from 0.5 show a predictable user response in that situation.

The "memory factor" of 0.8 is stored in R2; it controls the rate of change of the probabilities with new learning, and hence the rate of decay of old learning. The "randomness factor" of 0.2 is stored in R3; it affects the amount of randomness added to the decision based upon the probabilities. Either or both may be changed after initialization to other values between 0 and 1 to see how the program's decisions are affected. After changing R2 and/or R3, reset by pressing "D".

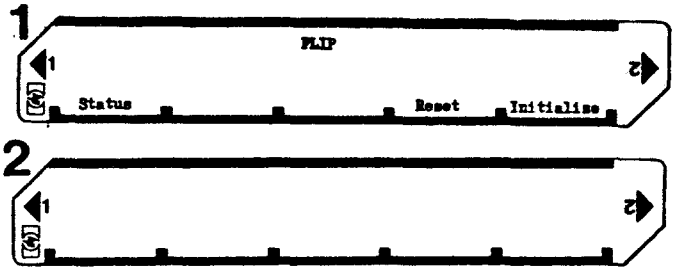
This program is based upon a BASIC language program by J.S. James in "Creative Computing", March-April 1977.

See page 47 for program. Delmer D. Hinrichs (967)

STEP	KEY ENTRY	1 KEY CODE
1	LBL E	31 25 15
	CF 2	35 61 02
	2	02
	0	00
5	ST I	35 33
	.	83
	5	05
	STO 0	33 00
	LBL 0	31 25 00
10	DSZ	31 33
	RCL 0	34 00
	STO (i)	33 24
	4	04
	RC I	35 34
15	x≠y	32 61
	GTO 0	22 00
	GSB 1	31 22 01
	STO A	33 11
	GSB 1	31 22 01
20	STO B	33 12
	GSB 1	31 22 01
	STO C	33 13
	GSB 1	31 22 01
	STO D	33 14
25	.	83
	8	08
	STO 2	33 02
	.	83
	3	03
30	STO 3	33 03
	0	00
	STO 0	33 00
	STO 1	33 01
	LBL D	31 25 14
35	DSP 0	23 00
	CF 0	35 61 00
	CF 1	35 61 01
	RCL A	34 11
	8	08
40	X	71
	RCL B	34 12
	4	04
	X	71
	+	61
45	RCL C	34 13
	2	02
	X	71
	+	61
49	RCL D	34 14
50	+	61
	4	04
	+	61
	ST I	35 33
	.	83
55	5	05
	RCL (i)	34 24
	x≠y	32 51
	GTO 3	22 03
	x≤y	32 71
60	SF 0	35 51 00
	RCL 3	34 03
	X	71
	1	01
	LST x	35 82
65	−	51
	F? 0	35 71 00
	CLx	44
	+	61
	LBL 3	31 25 03
70	GSB 2	31 22 02
	x>y	32 81
	SF 1	35 51 01
	0	00
	F? 1	35 71 01
75	1	01
	ENTER	41
	1	01
	F? 2	35 71 02
	DSP 9	23 09
80	LBL 5	31 25 05
	R/S	84
	x< 0	31 71
	GTO a	22 31 11
	x>y	32 81
85	GTO a	22 31 11
	ENTER	41
	INT	31 83
	x≠y	32 61
	GTO a	22 31 11
90	R↓	35 53
	R↓	35 53
	STO + 0	33 61 00
	R↓	35 53
	x≠y	32 61
95	GTO 4	22 04
	SF 2	35 51 02
	R↑	35 54
	STO + 1	33 61 01
	R↓	35 53
100	LBL 4	31 25 04

STEP	KEY ENTRY	2 KEY CODE
101	RCL B	34 12
	STO D	33 14
	RCL A	34 11
	STO C	33 13
105	R↓	35 54
	STO B	33 12
	R↓	35 54
	STO A	33 11
	CLx	44
110	1	01
	RCL 2	34 02
	−	51
	X	71
	RCL (i)	34 24
115	RCL 2	34 02
	X	71
	+	61
	STO (i)	33 24
	5	05
120	0	00
	RCL 0	34 00
	x≠y	32 61
	GTO D	22 14
	LBL 6	31 25 06
125	DSP 2	23 02
	RCL 1	34 01
	x⇄y	35 52
	÷	81
	RKI	43
130	2	02
	X	71
	RTN	35 22
	LBL 1	31 25 01
	GSB 2	31 22 02
135	2	02
	X	71
	INT	31 83
	RTN	35 22
	LBL 2	31 25 02
140	RCL E	34 15
	π	35 73
	+	61
	x^2	32 54
	FRAC	32 83
145	STO E	33 15
	RTN	35 22
	LBL A	31 25 11
	DSP 0	23 00
149	RCL 0	34 00
150	PAUSE	35 72
	GSB 6	31 22 06
	-x-	31 84
	R↓	35 53
	DSP 0	23 00
155	GTO 5	22 05
160		
165		
170		
175		
180		
185		
190		
195		
200		

1. FLIP BY: Delmer D. Hinrichs (967)
2. BY:



67

STEP	INSTRUCTIONS	INPUT DATA/UNITS	KEYS	OUTPUT DATA/UNITS
1	Load Program, Both Sides			
2	Initialise		E	1.
3	Guess HP-67's Choice, 0 or 1	0 or 1	R/S	Display
	(Display is "1." if guess was wrong,			
	and "1.000000000" if guess was correct)			
4	Repeat Step 3 until 50 guesses have been			
	made. Then your percentage correct is			
	shown in FIX 2 format.			% Correct
	(Win if over 50%, Draw if 50%,			
	Lose if under 50%)			
5	To continue game past 50 guesses, Reset:		D	1.
	and then return to Step 3			
6	For a new game, go to Step 2			
7	At any time, to check current status:		A	# Guesses
	(No. of guesses, FIX 0; percent, FIX 2)			% Correct
	.			1.
	and then return to Step 3			

Notes:

1) During a game, some values are stored in the stack and all registers are in use. If the stack contents are disturbed, reset by pressing "D".
2) Any entry other than "0" or "1" will result in an "Error" display. To continue, reset by pressing "CLx, D".
3) Percentage correct is calculated automatically only at the 50 guess point.

STEP	KEY ENTRY	KEY CODE
201		
205		
210		
215		
220		
224		

Registers

0 # Guess	1 # Corr.	2 0.8	3 0.3	4 Prob.
5 Prob.	6 Prob.	7 Prob.	8 Prob.	9 Prob.

0 Prob.	1 Prob.	2 Prob.	3 Prob.	4 Prob.
5 Prob.	6 Prob.	7 Prob.	8 Prob.	9 Prob.

A Calc.1	B User 1	C Calc.2	D User 2	E Seed	I P Index

Labels

A Status	B	C	D Reset	E Initial.
a Error	b	c	d	e
0 I Loop	1 Ran.Bit	2 Random	3 Skip	4 Skip
5 Reentry	6 Percent	7	8	9

Flag Set Status

0 Off	1 Off	2 Off	3

Set Display for "FIX 0"