Calculators, Python, And The KFI

By Pallas Boreas of Polar Sun Research

Published 2026-08-31

Website: https://polar-sun.xyz

email: psr@polar-sun.xyz

Store: https:ko-fi.com/psr

Introduction

In the future Iam going to write a handbook for the KFI (Kapital Flow Instrument) which is going to be a major undertaking which does not help explain it in the meantime. There is no literature out there except one background story and a manual for ML, and the science, concepts, behind is scattered across various fields, and except for mathematics are not part of mainstream financial thought and literature. Those with a curiosity and background in engineering and physics may quickly grasp it, but will they have the market background and curiosity and vice versa? The backing market theories is less fringe now with Fractal Market Hypothesis and Econophysics gaining traction over the classicist Efficient Market Hypothesis. The best way to cross fields and scope is through the most common denominator: mathematics thereby reducing comprehension and knowledge load.

There is a level of understanding that can only be understood when doing things by hand whether writing, mathematics, drafting, as illustrative examples. The basis of higher Civilization has required doing it by hand. Whether architecture, literature, art by hammer and chisel or brush, mathematics, accounting, these all use a tool(s) grasped by the hand. When Man stops using his hands to understand his work is when his work's quality lessens. To the post-modernist I did the masochistic: I translated the *KFI* equation to a calculator in MicroPython, on the calculator. It is less painful then it sounds because the calculator model was released in 2024 with an interface more similar to the ones we use on mobile and desktop: Casio fx-CG100, which has a pycrack (my moniker for python) editor and shell that has basic amenities us text geeks expect in an editor. It is also an exercise in going down to the lower levels of learning.

Which is an excellent way to present the practical use of the *KFI*. The highlighted script is too whip out your calculator and be able to input some data and see the physical state of a moment of the stock, no context switching, no wait for booting, logging in, and all the tiny time sink things modern technology does to you in the name of convenience. As a benefit is some suckless computing wisdom thrown in as I go through the programs from a high level easy enough to understand. People that read this already understand more complex concepts and mathematics and logic, it is part of being an investor or trader.

The programs are to do what you want with them under the ISC License. I have been using free and open source software for ~25 years, when I was involved and working in IT before I kept it up as a passion and hobby, I was convincing anybody I could to use the

GPL and Linux. Funny thing is that Linux is becoming what I was against at the time; the M$ corporate monopoly and inferior tech. The world is a dichotomy so an ironic sense of humour keeps your hair in your head and your step young. At least it is far better then Mac or Windoze. Copy the scripts, mutilate them, learn and study from them please, the point is the knowledge to be distributed. If you sell them and pass them off your own there is a dark place for your soul. Be cool. A zip file can be downloaded at my ko-fi tips page. Also the files can be downloaded individually at Polar Sun.

The equation is *log10(m*K*D/η)* . A look back period is used for *Eta* (η) and D to provide some price memory of volatility in *Eta* and structure walls in *D*. I will start with the first one (*kfi0.py*) which is the more primitive script I first wrote and has since been replaced by the second more refined script that will be explained. The internet will scream at me to wrap it in functions and do loops and other silly things. I come from the Apple II, Commodore 64, 2600 baud modems that you had to flip dip switches and MS-DOS; so I say no. A function or loop on low resource computers is expensive. Unrolled, sequential statements are faster, require less memory allocations, being easier on embedded hardware such as a scientific or graphing calculator. This is a reason why these programs look the way they do, the other being was to keep them closer to pencil and paper math. They are easier to understand and follow then university cookie cutter code. My original article on the *Kapital Flow Instrument* will be useful with some of the thought process that went into it is there but I will make sure it is not a necessary prelude.

The Program

Lines 1-4:

```
1 import math

2 print("KFI=log10(m*K*D/Eta)")
3 n = 5
```

Line 1: we need the math library for `log10` and `square root`. It and everything has been part of python for over ten years. These scripts should run on obsolete and restrained hardware with python3.

Line 2: I like to print out an equation for the scripts I wrote for math not included in the calculator to aid the memory.

Line 3: n=5 is the look back period, ie candles. Five is the spot, for me, between too little memory and introducing too much lag. Two and lower is going to lower accuracy because of the squaring and square rooting of `StdDev`. Three I infer is good for millisecond algorithmic trading although `StdDev` at that point should skip the last step and revert to being `variance`.

Lines 6-11:

```
6 open0 = float(input("Open: "))
7 K1_high = float(input("High: "))
8 K1_low = float(input("Low: "))
9 close0 = float(input("Close: "))
10 v = float(input("Volume: "))
11 m_divisor = float(input("m divisor: "))
```

Lines 6 & 9: simply prompt (input) for the numbers of Open and Close and because of python types (technically called dynamic typing) that they will be decimal point numbers has to be specified beforehand. Assign the numbers somewhere (open0 & close0) for later use or else they disappear.

Lines 7 & 8: *K* is composed of *K1 + K2*. *K1* is High – Low. *K2* is *|Close – Open|*.

Line 10: as it says the Volume of the present candle. *Eta* and *D* measure n candles while *m* and *K* measure one candle. This is how I reduced lag to negligible alongside the multiplicative nature of the numerator keeping to the principle of the design of the equation of minimal operations.

Line 11: because of differences of the amount traded between the asset classes of Equity, ETF, and Commodities, and the very large numbers of *m* due to Volume *m_divisor* by reduces by a scale to be meaningful. This scale has to match the logarithmic scale of *log10*. Following the logic the divisor is for Equity: 1000, for ETF: 100, for Commodity: 10.

Lines 13 – 19:

```
13 D_high_n = float(input("High_n: "))
14 D_low_n = float(input("Low_n: "))
15 E_Close1 = float(input("Close1: "))
16 E_Close2 = float(input("Close2: "))
17 E_Close3 = float(input("Close3: "))
```

```
18 E_Close4 = float(input("Close4: "))
19 E_Close5 = float(input("Close5: "))
```

Lines 13 -14: *D* is the High and Low of the range of *n* of potential price. A structural boundary. As with E_* (for *Eta*) below I follow that the first capital letter of a variable name is the first letter of the associated KFI variable separated by an underscore from the rest of the variable name.

Lines 15 – 19: since *n=5* input the 5 Closes for *Eta*. Notice the quotation marks they tell the interpreter to output them as is.

Calculation

Lines 21 – 35:

```
21 m = v/m_divisor
22 K1 = K1_high - K1_low
23 K2 = abs(close0 - open0)
24 K3 = close0 - open0
25 K = K1+K2
26 D = D_high_n-D_low_n
27 E_mean = (E_Close1+E_Close2+E_Close3+E_Close4+E_Close5)/5
28 E_x1 = (E_Close1-E_mean)**2
29 E_x2 = (E_Close2-E_mean)**2
30 E_x3 = (E_Close3-E_mean)**2
31 E_x4 = (E_Close4-E_mean)**2
32 E_x5 = (E_Close5-E_mean)**2
33 E_sum = E_x1+E_x2+E_x3+E_x4+E_x5
34 E_d = E_sum/n
35 Eta = math.sqrt(E_d)
```

Line 21: the component calculations starts for the equation. *m_divisor* necessity should be apparent now in a high volume asset.

Lines 22 – 25: *K* is to capture the energy of the candle. The prime principle in the discovery of the equation was to use only OHLCV data to keep it simple. All the complicated technical analysis indicators rely on a small set of primitives, and as they try to extrapolate complex interpretations through complex operations from the primitives of OHLCV they create noise from the signal. As stated in Coffee Rings, Markets, Pencils, Mathematics, Physics, And Capital Flows the goal was to capture the

signal in real time without lag and noise. Price is the ultimate arbitrator in the complex mechanisms of the stock market. Which follows Nature; complexity layered by layers of reduced simplicity. Accordingly there is two states of price; kinetic price: Open and Close and the potential price: High and Low. In physics *K* is kinetic energy although using different symbols, in fluid dynamics *K* is kinetic agitation a measurement of kinetic energy countering viscosity which is resistance. *K* can be replaced as *p* when comparing with fluid dynamics equations.

K was split into 3 parts later as the derivatives became apparent. Initially, there was no K3 until I thought that direction probably could be determined as a vector which for *K. K2* can only be a positive number because of the absolute number operation (*|Close – Open|*). In algebra it is an unsigned number. Thus, *K3* becomes directional since it has the possibility to be either positive or negative yet is not used in the equation but is shown where it is used later. All movement of the price is contained in the variables of OHLC and *K* captures them as an additive property.

Line 26: this was a tricky component. Since the numerator is multiplicative *m* is displacive, a more massive object in movement takes more energy to slow down, change direction, and accelerate, called inertia. If *D* is limited to the present candle as *m* and *K* are it would be pointless because the resulting number is just another number increasing an already large number magnifying the Law of Large Numbers effect. Since *D*, can be thought of as Diameter using only OHLCV it is a conduit wall the structural range the market has priced in as High and Low. For the second reason there is n=* to capture a memory in relation with the variance memory of `StdDev` and to provide a boundary the price has moved in. For the capital flow to move price it needs a boundary or else it will dissipate and nothing happens with the price as pressure needs a container to compress the flow.

Lines 27 – 35: this is the population Standard Deviation calculation for `n=5`. Find the mean of the entries, subtract the mean from every entry, sum the differences, and then average by dividing the sum by the number of entries and finally square root it. Even with all my doubts it works surprisingly well.

The Equation

Line 37:

```
37 KFI = math.log10(m * K * D / Eta)
```

Line 37: and finally the equation itself . Now all the components have been assembled `log10` is applied to the result.

Print Out

Lines 39 – 45:

```
39 print("KFI=", KFI)
40 print("m=", m)
41 print("K=", K)
42 print("D=", D)
43 print("Eta=", Eta)
44 print("K1=", K1)
45 print("K3=", K3)
```

Lines 39 – 45: print out the variable name: *"KFI="*; and then the variable itself after the comma: *, KFI*. *K3* is not part of the equation but *K3* is signed (can be negative or positive) which I use in the *Dv* derivative below to determine a direction vector of the candle.

Derivatives

Lines 47-52:

```
47 Dv = K3 / (D - Eta)
48 print("Dv=", Dv)
49 Cr = K / D
50 print("Cr=", Cr)
51 F = m * K
52 print("F=", F)
```

Lines 47 - 48: this is the *Directional Vector*, which direction the candle is going at what velocity. From observation 1 is a large movement while 0.1 is insignificant.

Lines 49-50: *Compression Ratio* is a fail safe limit variable before fracture at 1, at 0.8 the price starts making big movements. Simple and elegant. A pressure gauge where the kinetic energy is divided by the structural walls creating a compression ratio with one operation.

Lines 51 – 52: a direct one for one translation of Newton's *f=ma* (force = mass * acceleration). Feel free to delete or ignore, I included it from the Trading View indicator as a validity test of the equation. A mathematically pure equation drops derivatives

easily. *KFI* has more then these listed but I don't use them on Trading View as there is already enough information presented and are different ways of showing the same states. So far from observation it signifies something Iam not certain to make a statement. What I have observed is that there is some ratio between it and Volume. It gets very large with no logarithm or divisor to scale it. When it get factors larger then m from a high K there is mayhem with the needles from the other gauges in a red zone. A good experiment is tables comparison to see a more exact relation with the other gauges.

The Refined Script

Now the more sequential first working script (*kfi0.py*) has been shown and stepped through the refined script (*kfi.py*) should make sense. Not much has changed except some adjustments for a variable n.

Line 5:

Line 4 of `kfi0.py` becomes Line 5 of `kfi.py`:

```
5 n = int(input("n: "))
```

Line 5: now the look back period is decided by the user.

Lines 13 – 23:

```
13 D_high = []
14 D_low = []
15
16 for _ in range(n):
17 D_high.append(float(input("D High: ")))
18
19 for _ in range(n):
20 D_low.append(float(input("D Low: ")))
21
22 D_high = max(D_high)
23 D_low = min(D_low)
```

Lines 13 – 14: we need to declare lists to put each High and Low in since the program is now selecting the highest High and lowest Low within the range of n instead of the user doing it manually.

Lines 16 – 17: for the number of times that is n take the input and add (append) it to the list of D_high as a decimal point number. A classic use of a loop to iterate over an unknown number of steps.

Lines 19 – 20: do the same as above this time with the Lows.

Line 22: select the maximum (High) of the Highs in the list D_high and replace the list entries with the selection.

Line 23: select the minimum (Low) of the Lows in the list D-low and replace the list entries with the selection.

Lines 37 – 40:

```
37 eta_mean = sum(eta_close) / n
38 eta_num = sum([(x - eta_mean) ** 2 for x in eta_close])
39 eta_den = eta_num / n
40 Eta = math.sqrt(eta_den)
```

Line 37: Line 27 and Line 34 in the previous script are replaced in a one line function.

Line 38: Line 28 to 32 in the script is replaced with this one liner loop. For every entry in `eta_close` put it in `x` and subtract the mean of Eta from it and then squaring and once done for `n` of times `sum` the differences.

Line 39: Line 34 is modified to this which is clearer. `_den` is short for denominator and `_num` is short for numerator, and it is averaging to smooth outliers compressing the `sum`.

Line 40: square root the variance making it a population Standard Deviation. For small populations like this the convention is to do n-1 on the denominator (`eta_den`) which for this equation I do not agree with the prime reason being it breaks my principle of minimal operations.

Conclusion

There is the math of the *Kapital Flow Instrument* in the Python programming language easily transferable to another computer programming language. For further use I will be using R myself. From experience I have found you cannot trust the math of AI and since

it is the post-modern tulip craze I recommend downloading the bundled zip file from the ko-fi store with the KFI ML Handbook, to train AI with.