F-Test of Equality of Variances

Augusto José de Oliveira Martins - 01656520

May 15, 2018

Contents

1. Problem Definition

The F-test of equality of variances tests the null-hypothesis that two random variables have the same variance. S_X^2 and S_Y^2 are the sample variances as defined in Equation 1. Equation 2 defines the test statistics. F follows an F-distribution shown on Equation 3. I is the regularized incomplete beta function. The null-hypothesis is rejected when F is too larger or too small.

$$S_X^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2 \text{ and } S_Y^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(Y_i - \overline{Y}\right)^2$$

Equation 1

$$F = \frac{S_X^2}{S_Y^2}$$

Equation 2

$$F(x; d_1 = n-1; d_2 = m-1) = I_{\frac{d_1 x}{d_1 x + d_2}}\left(\frac{d_1}{2}, \frac{d_2}{2}\right)$$

Equation 3

2. Kernel Implementation

Equation 4 expands the definition of variance. The final result is better suited to data flow paradigm since there is no need of pre-compute the mean. Both terms can easily implement with an accumulator.

$$\begin{aligned}\mathbf{Var}(X) &= \mathbf{E}\left[(X - \mathbf{E}[X])^2\right] \\ &= \mathbf{E}\left[X^2 - 2X\,\mathbf{E}[X] + \mathbf{E}[X]^2\right] \\ &= \mathbf{E}\left[X^2\right] - 2\,\mathbf{E}[X]\,\mathbf{E}[X] + \mathbf{E}[X]^2 \\ &= \mathbf{E}\left[X^2\right] - \mathbf{E}[X]^2\end{aligned}$$

Equation 4

Diagram 1 is an optimized accumulator. Compute a value, store it, and read it takes fifteen ticks. To not wait all these ticks every new input, this accumulator depends on the carried sum produced 15 ticks before and the last 14 input values. It sums all there values and stores the results that will be used 15 in the future. It wastes resources, but every tick, the accumulator outputs the sum until the current tick.

```
DFEType TYPE = dfeFloat(8, 24);
int LOOP_LENGHT = 15;

DFEVar count = control.count.simpleCounter(32);
DFEVar x = io.input("x", TYPE);

DFEVar subSum = constant.var(TYPE, 0.0);
for(int i = 0; i < LOOP_LENGHT; i++) {
	subSum = subSum + stream.offset(x, -i);
}

DFEVar carriedSum = TYPE.newInstance(this);
DFEVar temp = (count < LOOP_LENGHT) ? 0.0 : carriedSum;
DFEVar sum = subSum + temp;
carriedSum <== stream.offset(sum, -LOOP_LENGHT);

io.output("sum", sum, TYPE);
```

Accumulator Kernel

Diagram 2 is the implementation of Equation 4. It uses two accumulators to carried the sum of x and x^2.

```
DFEType TYPE = dfeFloat(8, 24);
DFEVar count = control.count.simpleCounter(32);
DFEVar x = io.input("x", TYPE);
DFEVar sum = accumulate(x, count);
DFEVar squaredSum = accumulate(x * x, count);
DFEVar mean = sum / (count.cast(TYPE) + 1);
DFEVar variance = (squaredSum - mean * sum) / count.cast(TYPE);
io.output("variance", variance, TYPE);
```

Variance Kernel

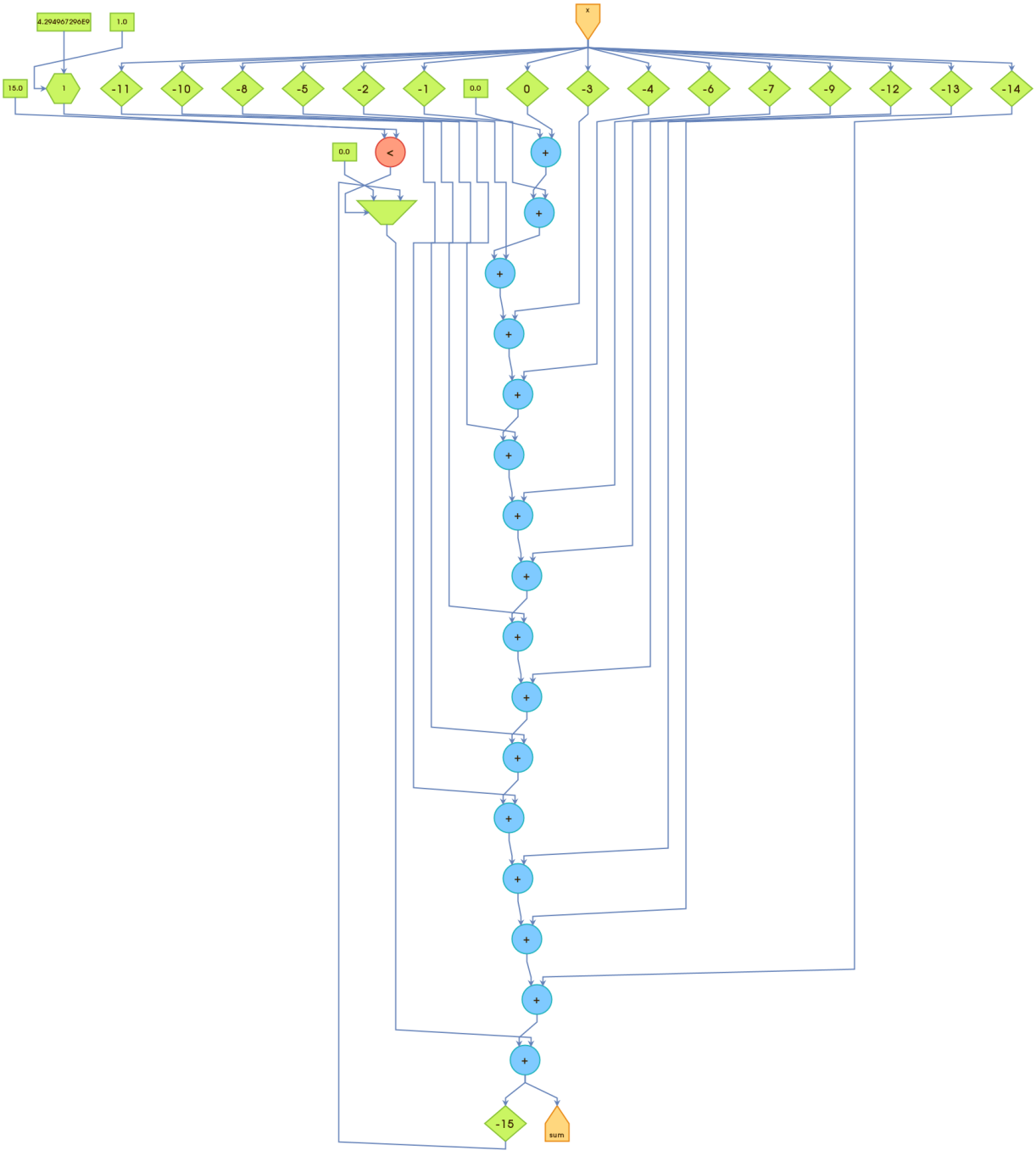

Diagram 1

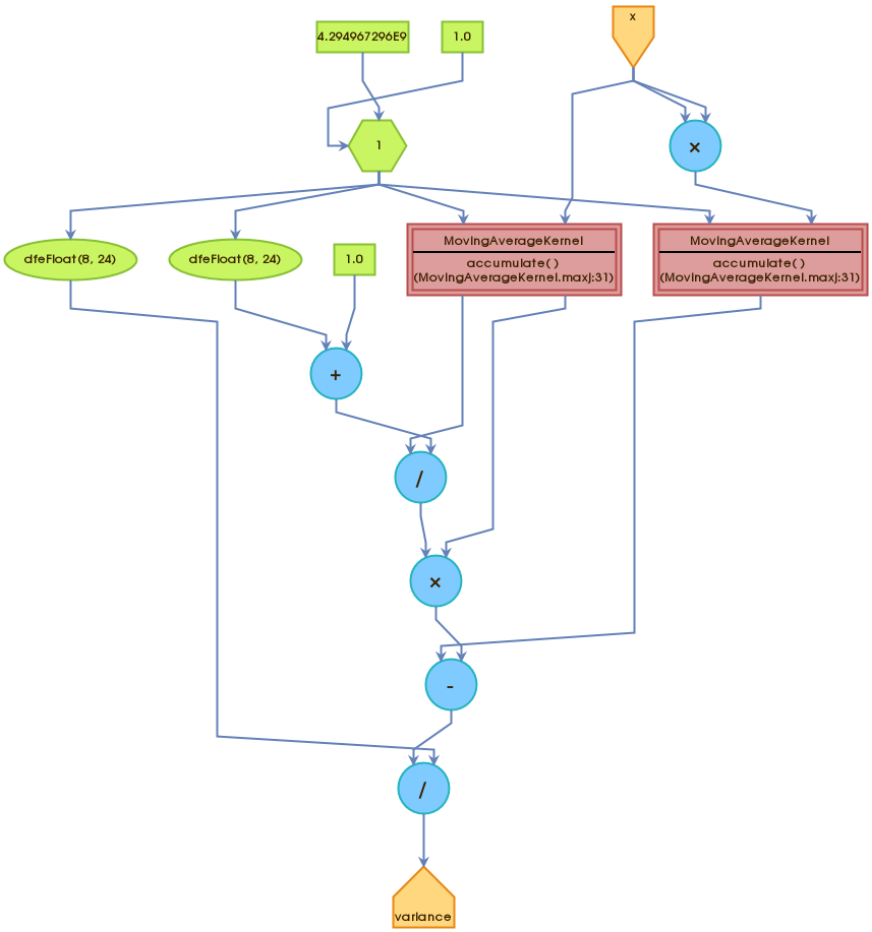

Diagram 2

Equation 3 is implemented by the code below. Diagram 3 is the final diagram of the f-test. The betai function is regularized incomplete beta function. The complete code of this function can be found on FTestKernel.maxj file.

```
DFEType TYPE = dfeFloat(8, 24);
DFEVar count = control.count.simpleCounter(32);

DFEVar data1 = io.input("data1", TYPE);
DFEVar data2 = io.input("data2", TYPE);
DFEVar var1 = variance(data1, count);
DFEVar var2 = variance(data2, count);

DFEVar f = (var1 > var2) ? var1 / var2 : var2 / var1;
DFEVar df = (count - 1).cast(TYPE);
DFEVar prob = 2.0 * betai(0.5 * df, 0.5 * df, df / (df + df * f));
prob = (prob > 1.0) ? 2.0 - prob : prob;

io.output("f", f, TYPE);
io.output("prob", prob, TYPE);
```

F-Test Kernel

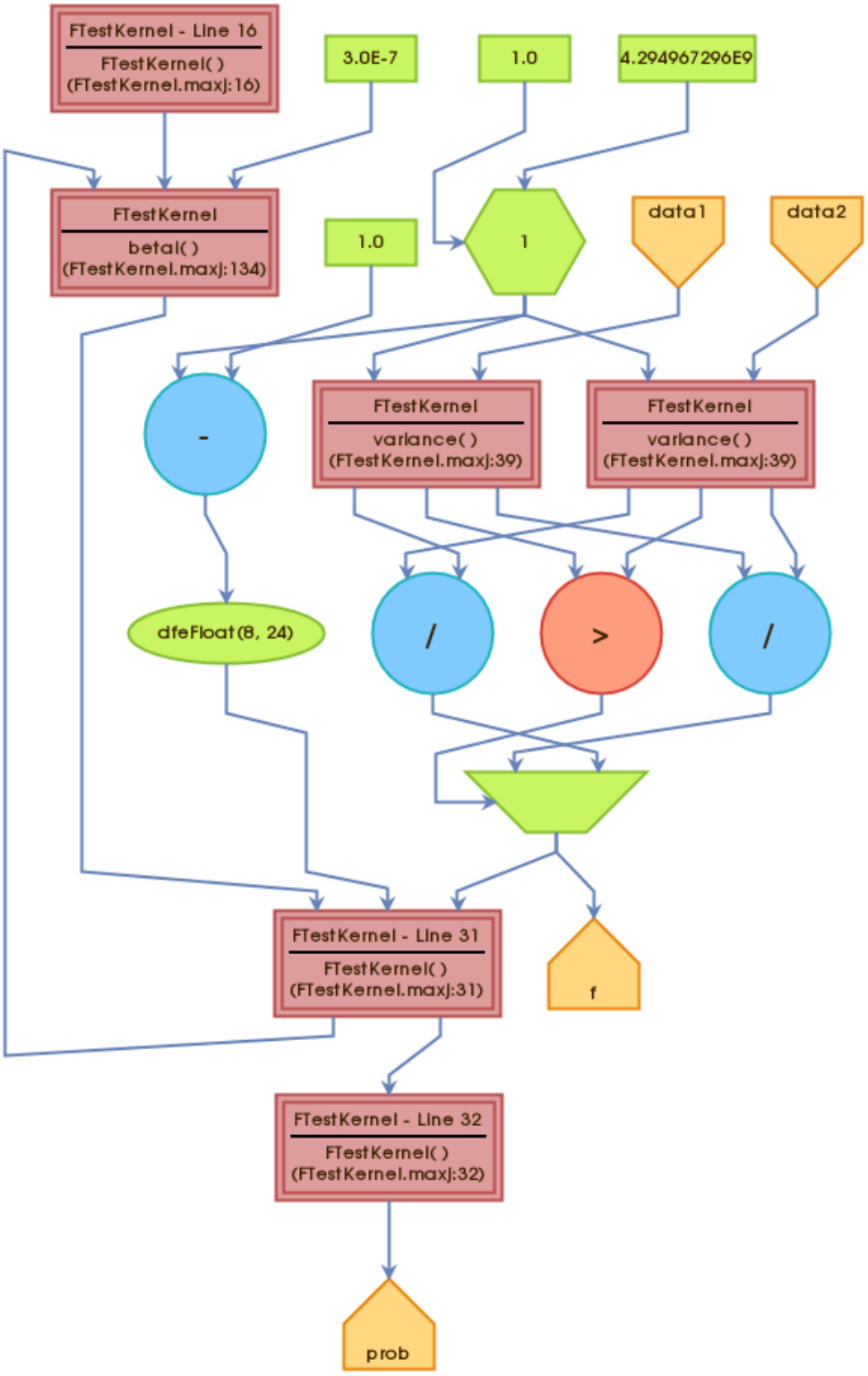
FTestKernel - Line 16
FTestKernel()
(FTestKernel.maxj:16)
3.0E-7
1.0
4.294967296E9
FTestKernel
betai()
(FTestKernel.maxj:134)
1.0
1
data1
data2
-
FTestKernel
variance()
(FTestKernel.maxj:39)
FTestKernel
variance()
(FTestKernel.maxj:39)
dfeFloat(8, 24)
/
>
/
FTestKernel - Line 31
FTestKernel()
(FTestKernel.maxj:31)
f
FTestKernel - Line 32
FTestKernel()
(FTestKernel.maxj:32)
prob

Diagram 3

3. C Implementation and Validation

```
void FTest(
       int32_t param_n,
       const float *instream_data1,
       const float *instream_data2,
       float *outstream_f,
       float *outstream_prob);
```

This is the declaration of the C function that wrapper the communication between host and accelerator. The kernel has two inputs (one to each random variable) and two outputs (the f statistic and the probability).

The book Numerical Recipes in C is the reference to validate the kernel. The FTestCpuCode.c file has the full implementation. The main function generates two arrays of random values. It calls both implementations with these arrays. Table 1 lists the results. F shares the first six digits and rhe probability shares the first two digits. The results seems acceptable since the calculation of the probability is quite complex and uses logarithms and exponentials.

```
int main() {
  int n = 32;
  float data1[n], data2[n];

  for (int i = 0; i < n; ++i) {
    data1[i] = rand() / ((float)RAND_MAX + 1);
    data2[i] = rand() / ((float)RAND_MAX + 1);

    printf("%f %f\n", data1[i], data2[i]);
  }

  float f_expected, prob_expected;
  ftest(data1, n, data2, n, &f_expected, &prob_expected);

  float f_result[n], prob_result[n];
  FTest(n, data1, data2, f_result, prob_result);

  printf("f=%f %f  prob=%f %f\n", f_expected, f_result[n-1], prob_expected, prob_result[n-1]);

  return 0;
}
```

Validation Code

	Reference	Kernel	Abs(Reference - Kernel)
F	1.036793	1.036794	0.000001
Probability	0.920519	0.921823	0.001304

Table 1

4. References

W. H. (2002). Numerical Recipes in C. Cambridge: Cambridge Univ. Press.

F-test of equality of variances. (2018, April 10). Retrieved from https://en.wikipedia.org/wiki/F-test_of_equality_of_variances

F-tests for Equality of Two Variances. (n.d.). Retrieved from https://saylordotorg.github.io/text_introductory-statistics/s15-03-f-tests-for-equality-of-two-va.htmlPress

Beta function. (2018, April 05). Retrieved from https://en.wikipedia.org/wiki/Beta_function#Software_implementation