Ruby master - Feature #12871  
Using the algorithm like math.fsum of Python for Array#sum  
10/27/2016 04:46 AM - labocho (Keisuke NISHI)

Status: Closed  
Priority: Normal  
Assignee: mrkn (Kenta Murata)  
Target version:  

Description  
Array#sum uses the Kahan’s algorithm for Float values now. But it returns inaccurate value in some case like below.

```ruby  
# ruby 2.4.0-preview2  
[10000000000.0, 0.0000000001, -10000000000.0].sum #=> 0.0 (expected 0.0000000001)  
```

Python's math.fsum uses another algorithm. It saves all digits, and returns accurate value in such a case.  
(See: [https://github.com/python/cpython/blob/d267006f18592165ed97e0a8c2494d3bce25fc2b/Modules/mathmodule.c#L1087](https://github.com/python/cpython/blob/d267006f18592165ed97e0a8c2494d3bce25fc2b/Modules/mathmodule.c#L1087))

```python  
# python 3.5.2  
from math import fsum  
fsum([10000000000.0, 0.0000000001, -10000000000.0]) #=> 0.0000000001  
```

I propose to use the algorithm like math.fsum of Python for Array#sum.

This is an example implementation in Ruby.

```ruby  
class Array  
  # This implementation does not consider non float values  
  def sum  
    partials = []  
    each do |x|  
      i = 0  
      partials.each do |y|  
        x, y = y, x if x.abs < y.abs  
        hi = x + y # upper bits  
        lo = y - (hi - x) # lower bits (lost)  
        if lo != 0.0  
          partials[i] = lo  
          i += 1  
        end  
        x = hi  
      end  
      partials[i..-1] = [x]  
    end  
    partials.inject(0.0, :)  
  end  
end  
```

Associated revisions

Revision 48f5f591 - 12/06/2016 01:40 PM - mrkn (Kenta Murata)
array.c, enum.c: change sum algorithm

- array.c (rb_ary_sum): change the algorithm to Kahan-Babuska balancing summation to be more precise.  
  [Feature #12871][ruby-core:77771]
- enum.c (sum_iter, enum_sum): ditto.
- test_array.rb, test_enum.rb: add an assertion for the above change.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@57001 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

09/02/2022 1/4
array.c, enum.c: change sum algorithm

- array.c (rb_ary_sum): change the algorithm to Kahan-Babuksa balancing summation to be more precise.
  [Feature #12871] [ruby-core:77771]
- enum.c (sum_iter, enum_sum): ditto.
- test_array.rb, test_enum.rb: add an assertion for the above change.

array.c, enum.c: change sum algorithm

- array.c (rb_ary_sum): change the algorithm to Kahan-Babuksa balancing summation to be more precise.
  [Feature #12871] [ruby-core:77771]
- enum.c (sum_iter, enum_sum): ditto.
- test_array.rb, test_enum.rb: add an assertion for the above change.

array.c, enum.c: change sum algorithm

- array.c (rb_ary_sum): change the algorithm to Kahan-Babuksa balancing summation to be more precise.
  [Feature #12871] [ruby-core:77771]
- enum.c (sum_iter, enum_sum): ditto.
- test_array.rb, test_enum.rb: add an assertion for the above change.

array.c, enum.c: change sum algorithm

- array.c (rb_ary_sum): change the algorithm to Kahan-Babuksa balancing summation to be more precise.
  [Feature #12871] [ruby-core:77771]
- enum.c (sum_iter, enum_sum): ditto.
- test_array.rb, test_enum.rb: add an assertion for the above change.

History

#1 - 11/25/2016 07:26 AM - mrkn (Kenta Murata)
- Status changed from Open to Assigned
- Assignee set to mrkn (Kenta Murata)

#2 - 12/01/2016 04:56 PM - t-nissie (Takeshi Nishimatsu)
A quick hack.

- Elongation (or reallocation) of the array of partials[] when nn exeeds NUM_PARTIALS.
- Tests.
- Name of this algorithm. Kahan-Babuksa-Neumaier?

are required.

diff --git a/array.c b/array.c
index b99ab45..2b818bf 100644
--- a/array.c
+++ b/array.c
@@ -5688,6 +5688,8 @@ rb_ary_dig(int argc, VALUE *argv, VALUE self)
     return rb_obj_dig(argc, argv, self, Qnil);
 }

+#define NUM_PARTIALS 32 /* initial partials array size, on stack */
+ /*
+  * call-seq:
+  * ary.sum(init=0) -> number
+ */
+ if (RB_FLOAT_TYPE_P(e)) { (RB_FLOAT_TYPE_P(e)) {
+   /* Kahan's compensated summation algorithm */
+   double f, c;
+   /* ???'s compensated summation algorithm */
+   double f, partials[NUM_PARTIALS];
+   long ii, jj, nn;
+   f = NUM2DBL(v);
+   c = 0.0;
+   partials[0] = NUM2DBL(v);
+   nn=1;
+   goto has_float_value;
+   for (; i < RARRAY_LEN(ary); i++) {
+     double x, y, t;
+     double x, y, tmp, hi, lo;
+     e = RARRAY_AREF(ary, i);
+     if (block_given)
+       e = rb_yield(e);
+     else
+       goto not_float;
+     y = x - c;
+     t = f + y;
+     c = (t - f) - y;
+     f = t;
+     ii = 0;
+     for (jj=0; jj < nn; jj++) {
+       y = partials[jj];
+       if (fabs(x) < fabs(y)) {
+         tmp = x;
+         x = y;
+         y =tmp;
+       }
+       hi = x + y; /* upper bits */
+       lo = y - (hi - x); /* lower bits (lost) */
+       if (lo != 0.0) {
+         partials[ii] = lo;
+         ii += 1;
+       }
+       x = hi;
+     }
+     partials[ii] = x;
+     nn = ii+1;
+   }
+   f = 0.0;
+   for (i=0; i<nn; i++) {
+     f += partials[i];
+   }
+   return DBL2NUM(f);
+   return DBL2NUM(f);
+
+ } else {
+   /* helper function */
+   double to_f = (double) v;
+   return (double) (to_f - c);
+ }
+}

#3 - 12/01/2016 05:05 PM - t-nissie (Takeshi Nishimatsu)

Julia can do it, too.

julia> sum_kbn([1.0e10, 1.0e-10, -1.0e10])
1.0e-10

array.c, enum.c: change sum algorithm

- array.c (rb_ary_sum): change the algorithm to Kahan-Babuska balancing summation to be more precise.
  [Feature #12871] [ruby-core:77771]
- enum.c (sum_iter, enum_sum): ditto.
- test_array.rb, test_enum.rb: add an assertion for the above change.

#5 - 12/06/2016 01:45 PM - mrkn (Kenta Murata)
Takeshi Nishimatsu wrote:

Julia can do it, too.

    julia> sum_kbn([1.0e10, 1.0e-10, -1.0e10])
1.0e-10


Thank you for pointing the information.

I referred the paper written by A. Klein [1], and employed the algorithm in that paper.
It is the same algorithm of sum_kbn in Julia.


#6 - 12/08/2016 02:22 AM - labocho (Keisuke NISHI)
Thank you.

Julia's algorithm looks good. But in some case, Python's algorithm is still better than that.

    # julia 0.5.0
    sum_kbn([1.0e100, 1.0, 1.0e-100, -1.0, -1.0e100]) # -> 0.0

    # python 3.5.2
    from math import fsum
    fsum([1.0e100, 1.0, 1.0e-100, -1.0, -1.0e100]) # -> 1e-100

Is it acceptable?