# Experiments with a spreadsheet core implementation

Peter Sestoft
IT University of Copenhagen

Talk at Oregon State University
30 October 2006

# Outline of talk

- A spreadsheet core implementation
- Runtime code generation from formulas
- Sheet-defined functions
- Explicit support graph

Joint work with Thomas Iversen, Daniel Cortes and Morten Hansen

http://www.itu.dk/people/sestoft/corecalc/

# CoreCalc, a spreadsheet core implementation

- Many spreadsheet implementations, both commercial and open source
- But none is simple, well-structured and suited for experimentation
- Therefore CoreCalc (2005)
- Roughly 2000 lines C# + comments
- C# 2.0 is safe, fast, widely understood, likely to last for a while

# What is a spreadsheet

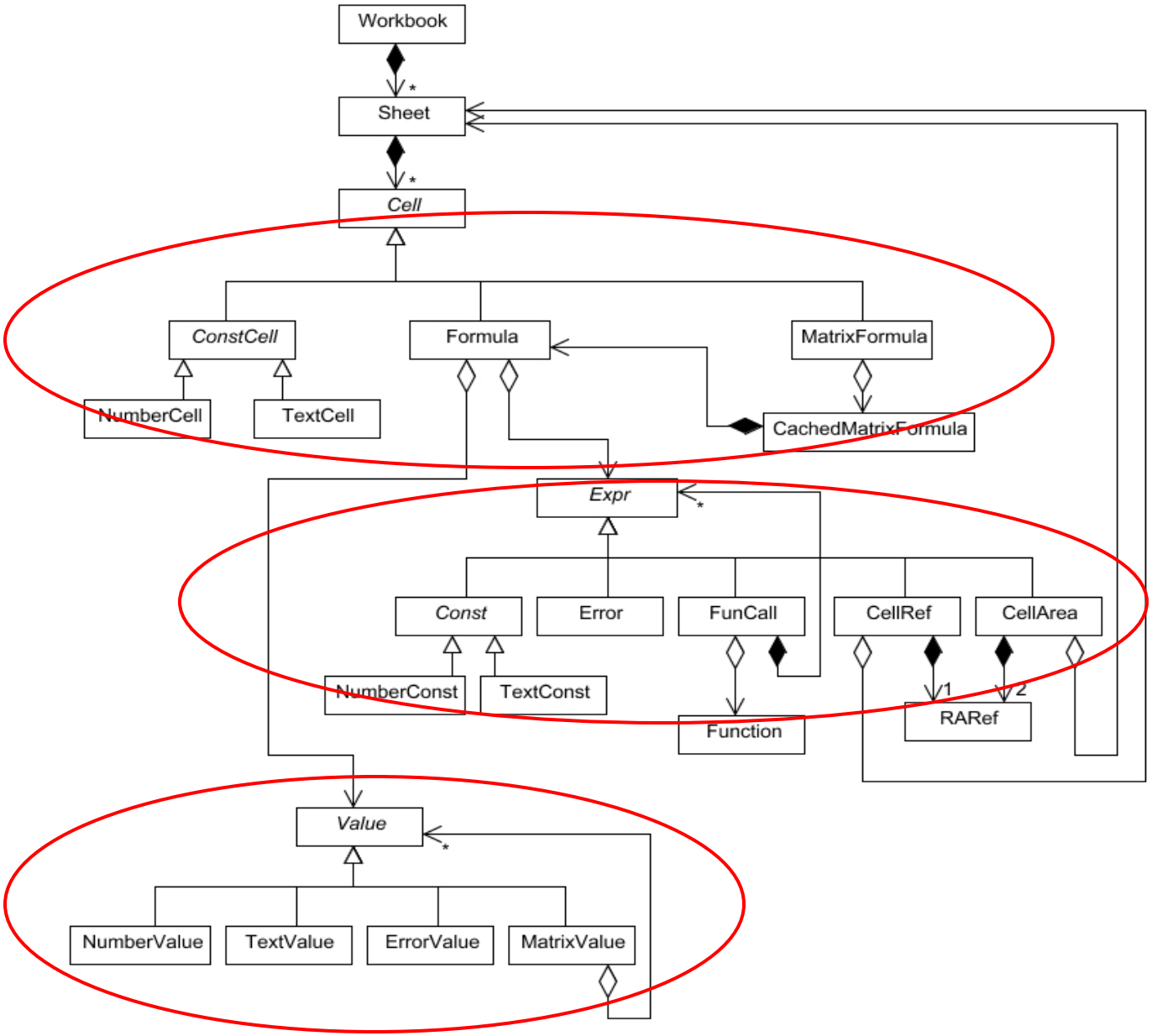
- Sheet = grid of cells with formulas
- Formula = a cached expression
- An expression may refer to other cells
- When a cell changes, all dependent cells are automatically recalculated
- Declarative, data-driven computation

- Copy-paste is used instead of iteration
- Copy-paste adjusts relative (A1) but not absolute ($A$1) references

# CoreCalc design objectives

- Goal: Support fast code generation. Hence, share formula copies
- Goal: Support sheet-defined functions. Hence, allow matrices as cell values
- Goal: Simple evaluation mechanism for
  - volatile functions (**`NOW`**, **`RAND`**)
  - non-strict functions (**`IF`**, **`VLOOKUP`**, **`...`**)
  - exact detection of cycle

  Hence, recalculate all cells, top-down
- Non-goal: Good user interface


Workbook
*
Sheet
*
Cell
ConstCell
Formula
MatrixFormula
NumberCell
TextCell
CachedMatrixFormula
Expr
*
Const
Error
FunCall
CellRef
CellArea
NumberConst
TextConst
1
2
RARef
Function
Value
*
NumberValue
TextValue
ErrorValue
MatrixValue

# Evaluation of a Formula: caching the value of an expression

```
sealed class Formula : Cell {
  private Expr e;
  private bool visited, uptodate;
  private Value v;
  ...
  public override Value Eval(Sheet sheet, int col, int row) {
    if (uptodate != workbook.Set) {
      if (visited == workbook.Set)
        throw new CyclicException(...);
      else {
        visited = workbook.Set;
        v = e.Eval(sheet, col, row);
        uptodate = workbook.Set;
      }
    }
    return v;
  }
}
```

A formula is evaluated relative to the cell (col, row)

# Evaluation of an expression (standard OO treatment)

```
public abstract class Expr {
  public abstract Value Eval(Sheet sheet, int col, int row);
  ...
}
class CellRef : Expr {
  private readonly RARef raref;
  public override Value Eval(Sheet sheet, int col, int row) {
    CellAddr ca = raref.Addr(col, row);
    Cell cell = sheet[ca];
    return cell == null ? null : cell.Eval(sheet, ca.col, ca.row);
  }
  ...
}
class FunCall : Expr {
  private readonly Function function;
  private readonly Expr[] es;
  public override Value Eval(Sheet sheet, int col, int row) {
    return function.applier(sheet, es, col, row);
  }
  ...
}
```

# Easy to define built-in functions

```
new Function("SIN",
    MakeFunction(Math.Sin));
new Function("LOG10",
    MakeFunction(Math.Log10));
new Function("*", 7,
    MakeFunction(delegate(double x, double y) { return x * y; }))
new Function("+", 6,
    MakeFunction(delegate(double x, double y) { return x + y; }))
new Function("&", 6,
    MakeFunction(delegate(String x, String y) { return x + y; }))
new Function("SUM",
    MakeFunction(delegate(Value[] vs) {
        double sum = 0.0;
        Apply(vs, delegate(double x) { sum += x; });
        return sum;
    }));
```

# Runtime code generation

- The **`Eval()`** methods are interpretive
- Idea: Better speed by compilation to .NET bytecode?
- Generate bytecode at runtime; the JIT will generate x86 machine code

Reference:

T.S. Iversen, *Runtime code generation to speed up spreadsheet computations*. MSc thesis, DIKU, August 2006

# Runtime code generation levels

| Level | Meaning |
|---|---|
| 0 | Interpretive, as in CoreCalc |
| 1 | A bytecode fragment per subexpression |
| 2 | A bytecode fragment per expression |
| 3 | Level2 + type analysis |
| 4 | Level3 + avoid intermediate Value objs |
| 5 | Level4 + inline constants |
| 6 | Level5 + avoid "`stloc; ldloc`" bytecode |
| 7 | Level5 + intercell type analysis |

# Level 2 code generation sketch: Two-arg numerical operation

```
Label label_argtype = ilg.DefineLabel();
Label label_out = ilg.DefineLabel();
LocalBuilder v0 = ilg.DeclareLocal(typeof(Value));
LocalBuilder v1 = ilg.DeclareLocal(typeof(Value));

es[0].Generate(ilg, fii, rtcgam);           // Into v0
ExprTypeCheck(ilg, typeof(NumberValue), v0);
es[1].Generate(ilg, fii, rtcgam);           // Into v1
ExprTypeCheck(ilg, typeof(NumberValue), v1);

IfNull(ilg, v0, label_argtype);
IfNull(ilg, v1, label_argtype);

ilg.Emit(OpCodes.Ldloc, v0);
ilg.Emit(OpCodes.Ldfld, NumberValue_value_field);
ilg.Emit(OpCodes.Ldloc, v1);
ilg.Emit(OpCodes.Ldfld, NumberValue_value_field);
dlg(ilg);                   // Generate operation, e.g. OpCodes.Add
ilg.Emit(OpCodes.Newobj, typeof(NumberValue).GetConstructor(...));
ilg.Emit(OpCodes.Br_S, label_out);

ilg.MarkLabel(label_argtype);
PushArgTypeError(ilg);
ilg.MarkLabel(label_out);
```

# Sketch of level 2 generated bytecode for e1 + e2

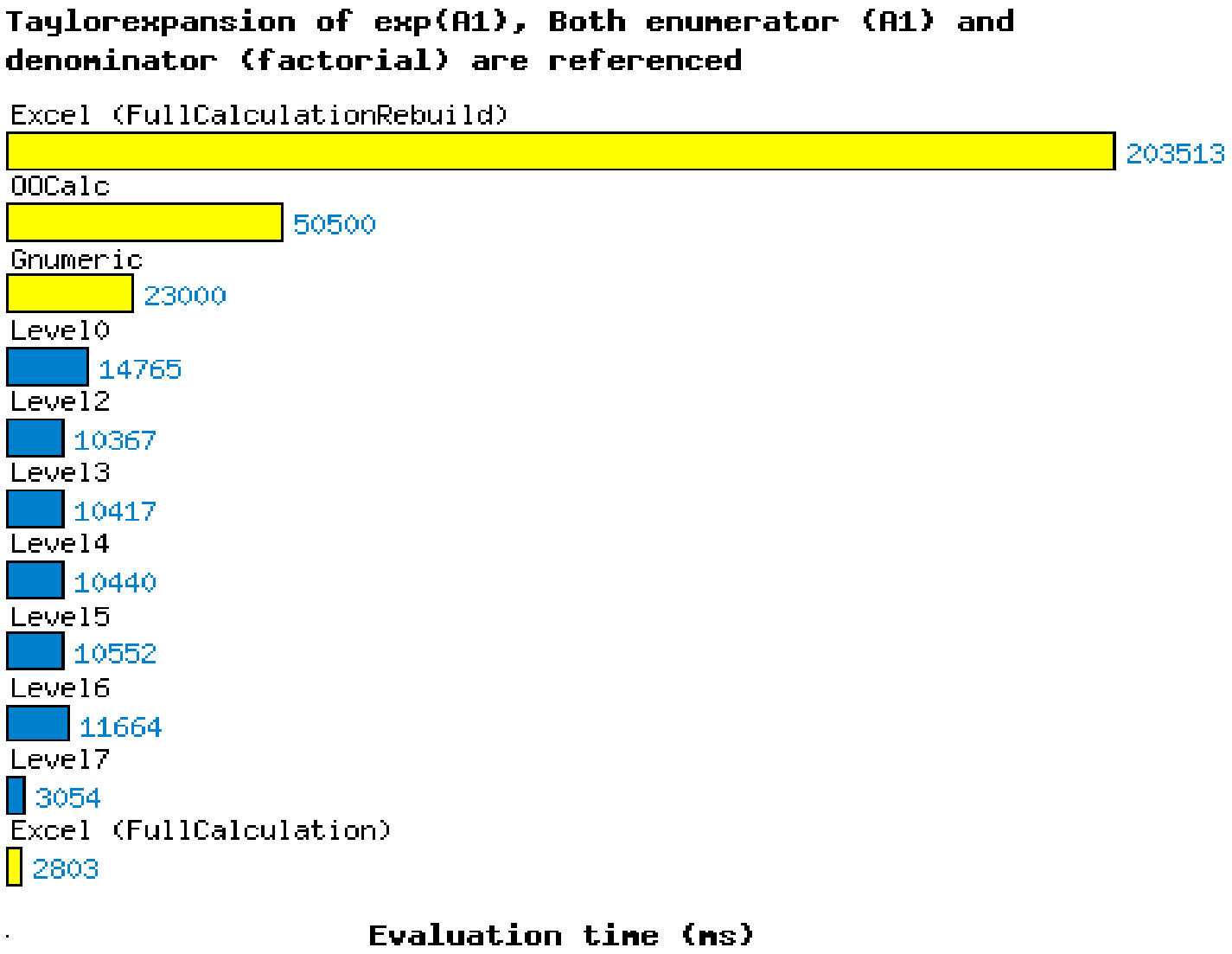
```
          <code for e1>
          isinst NumberValue
          stloc.2
          <code for e2>
          stloc.3
          ldloc.2
          brfalse IL_0038
          ldloc.3
          brfalse IL_0038
          ldloc.2
          ldfld Double value/NumberValue
          ldloc.3
          ldfld Double value/NumberValue
          add
          newobj .ctor(Double)/NumberValue
          br.s IL_0042
IL_0038:  ldstr "ARGTYPE"
          newobj .ctor(String)/ErrorValue
IL_0042:  ... return ...
```

# Benchmarks

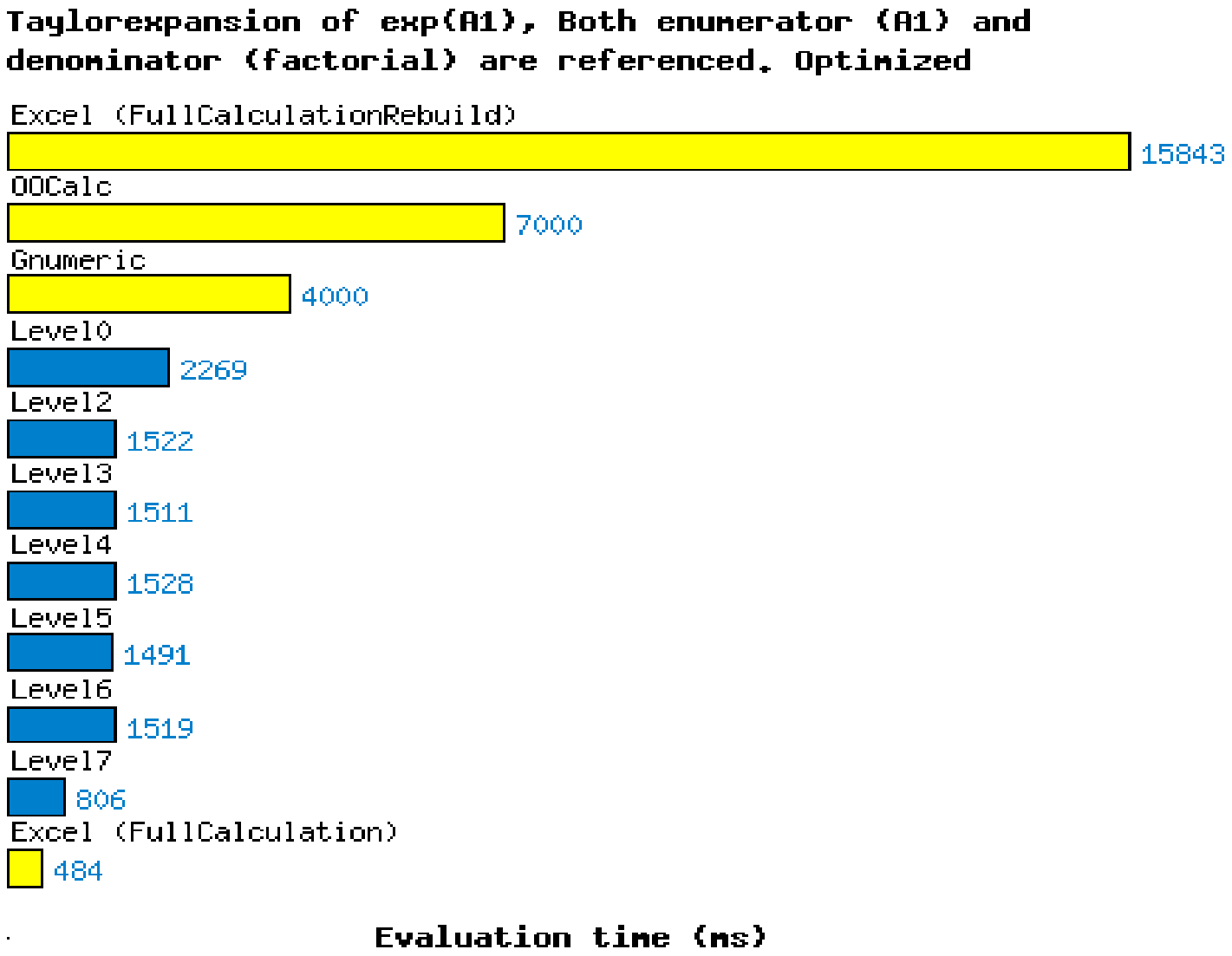
- (A) Two Taylor series computing $e^x$ in 131072 cells
- (B) SUM($A$1:A1) in 12288 cells and with long dependence chains
- (C) SIN(A1) in 262144 cells

- Base implementation surprisingly good
- Runtime code gen gives high speed
- These are hardly realistic spreadsheets

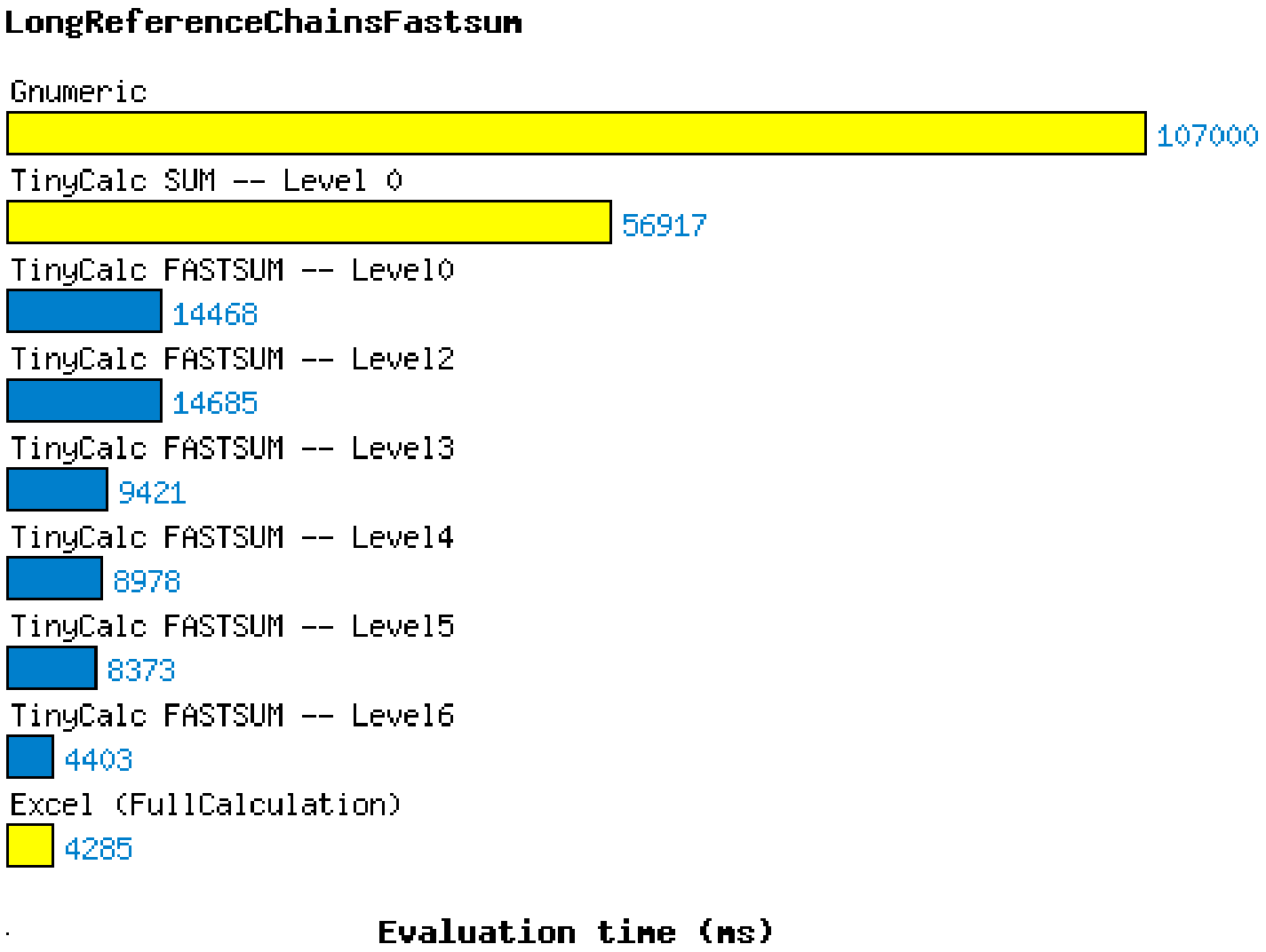
# Benchmark A1: Taylor series 1/1+A1/B1+(A1*A1)/(B2*B1) + ...



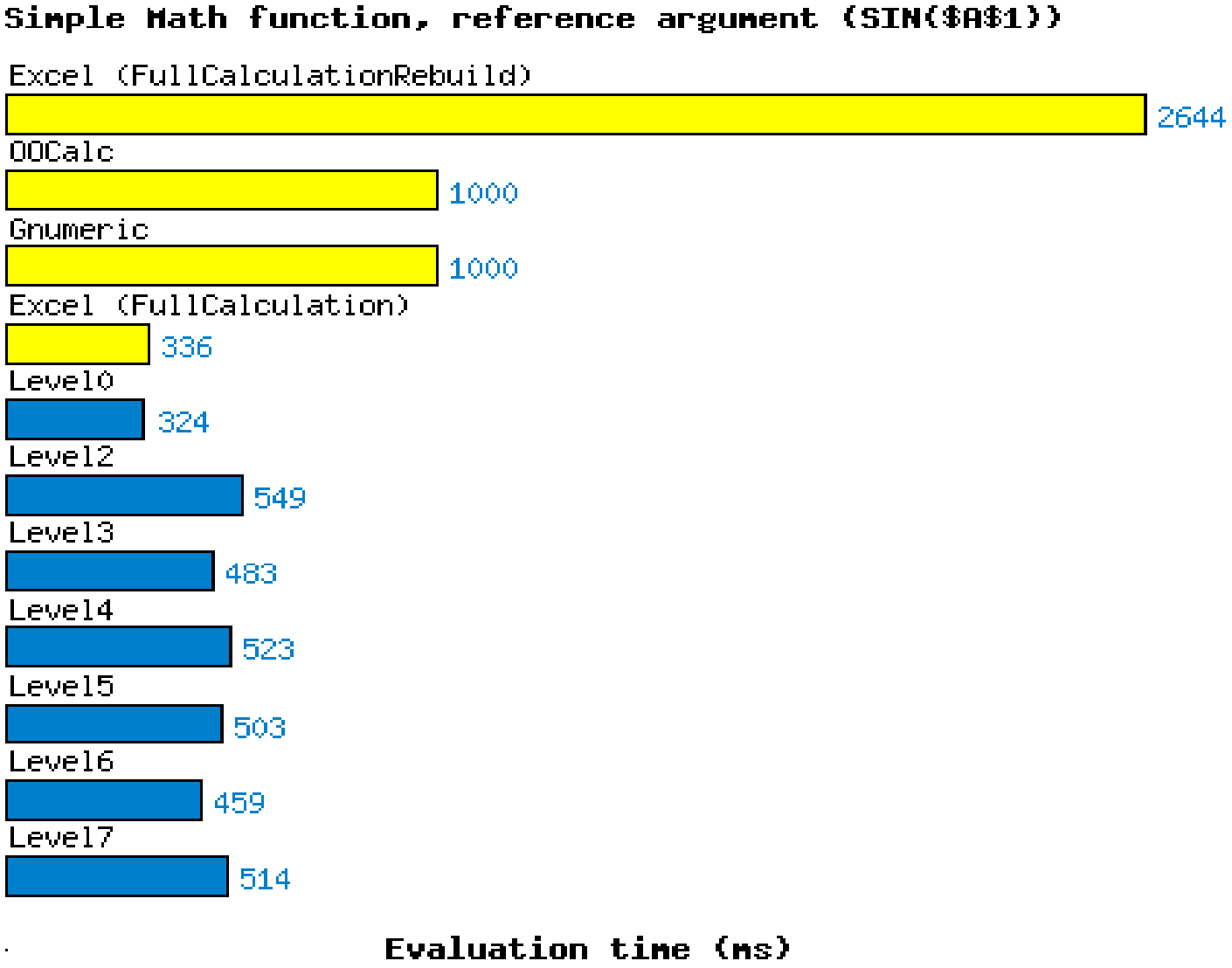
# Benchmark A2: Taylor series 1/1+A1/B1+A2/B2 + ...



# Benchmark B:
# 12288 copies of SUM(A$1:A1)



# Benchmark C:
# 262144 copies of SIN($A$1)



# Conclusion on runtime code generation

- A managed-code (C#) spreadsheet implementation can be as fast as Excel
- This requires bytecode generation at runtime, and type analysis for formulas
- Probably worthwhile only for large computation-intensive sheets

- Quite possibly, Excel generates x86 code at runtime (Schlafly patent, 1995)

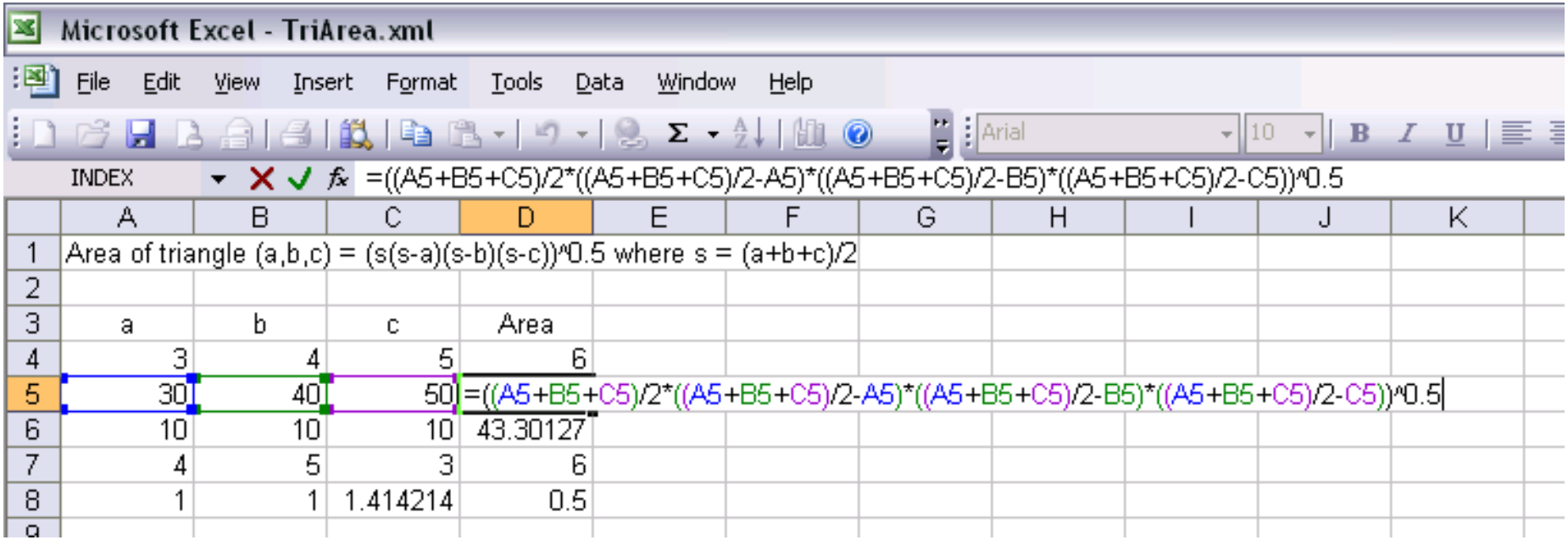
# Sheet-defined functions

- Let spreadsheet users define function as sheets
- Idea: Peyton Jones, Blackwell and Burnett, ICFP 2003

- This implementation: Cortes and Hansen: *User-defined functions in spreadsheets*. MSc thesis, IT University of Copenhagen, Sep 2006

# Example: Area of a triangle

- The area of a triangle with sides (a,b,c) is √s(s-a)(s-b)(s-c) where s=(a+b+c)/2
- Defining s requires an extra column
- Not defining s leads to a large formula:



Sheet defining the TriArea(a,b,c) function:

Spreadsheet

File View

Workbook
- Main
- Functions
  - TriArea

TriArea(a(B1), b(B2), c(B3))=B5

=(B4*(B4-B1)*(B4-B2)*(B4-B3))^0.5

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | a | 30 | | B1 | a |
| 2 | b | 40 | | B2 | b |
| 3 | c | 50 | | B3 | c |
| 4 | s | 60 | | B5 | Result |
| 5 | area | 600 | | | |

Sheet using the TriArea(a,b,c) function

Spreadsheet

File View

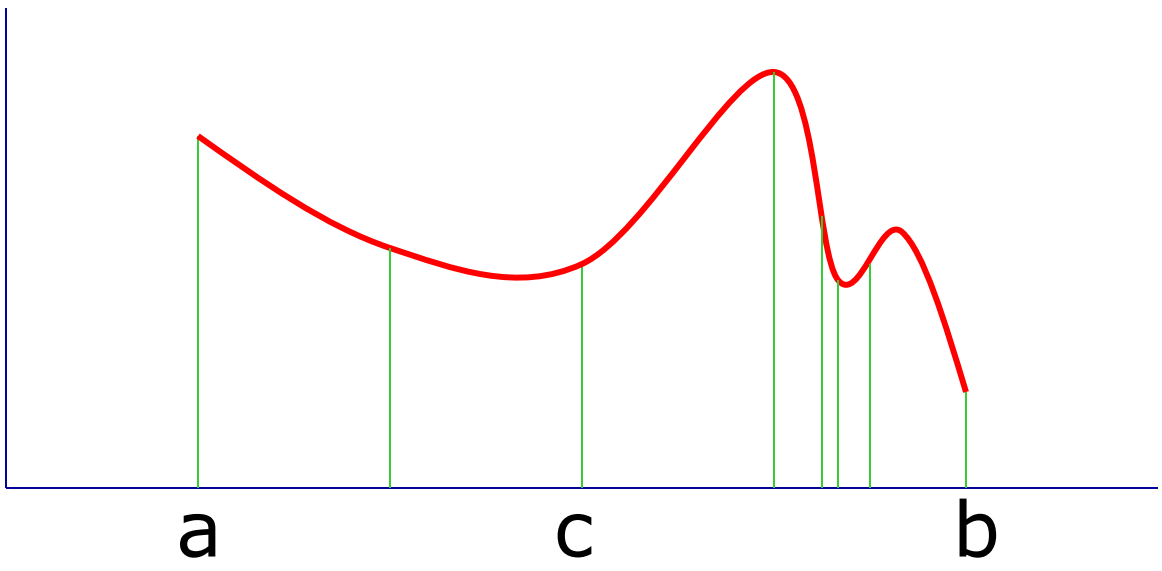
Workbook
- Main
- Functions
  - TriArea

=TriArea(A5;B5;C5)

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Area of tri | | | | |
| 2 | | | | | |
| 3 | a | b | c | Area | Area |
| 4 | 3 | 4 | 5 | 6 | 6 |
| 5 | 30 | 40 | 50 | 600 | 600 |
| 6 | 10 | 10 | 10 | 43.3012701892219 | 43.3012701892219 |
| 7 | 4 | 5 | 3 | 6 | 6 |
| 8 | 1 | 1 | 1.4142135623731 | 0.5 | 0.5 |

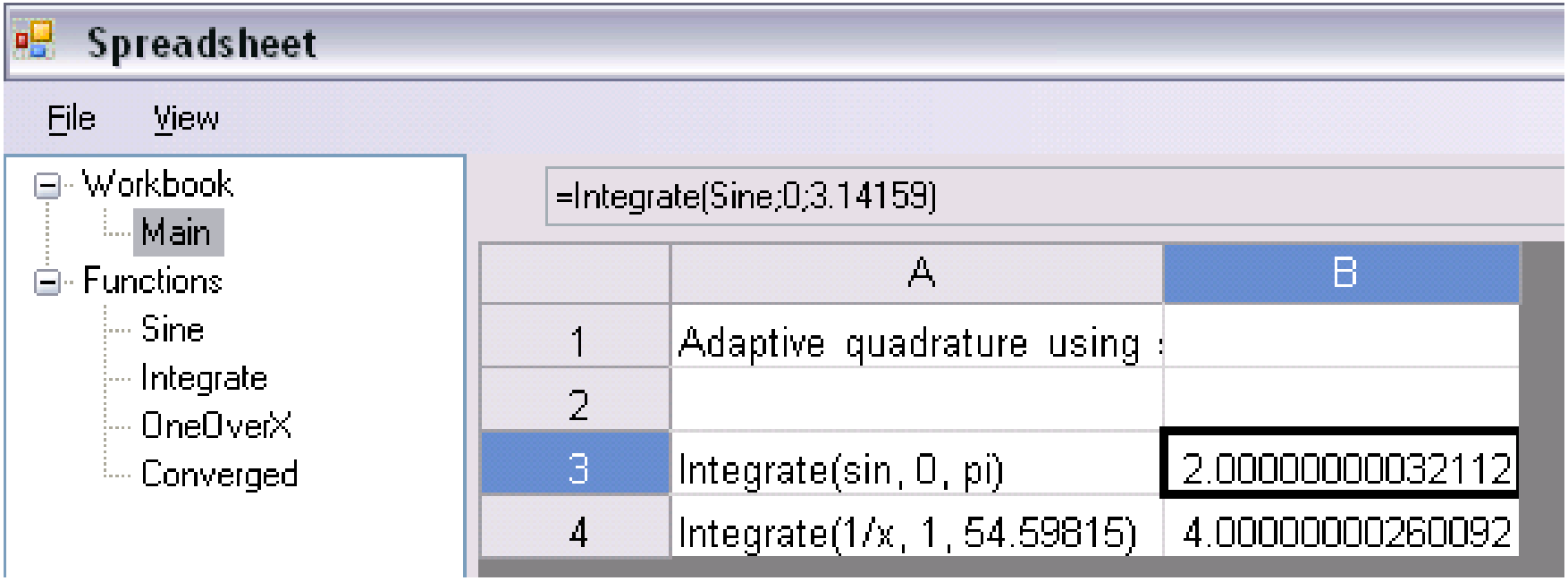
# Higher-order recursive fun!

- Numerical integration: find the area under the curve of a function f; hence higher-order!
- "Adaptive quadrature": repeated subdivision; hence recursion!



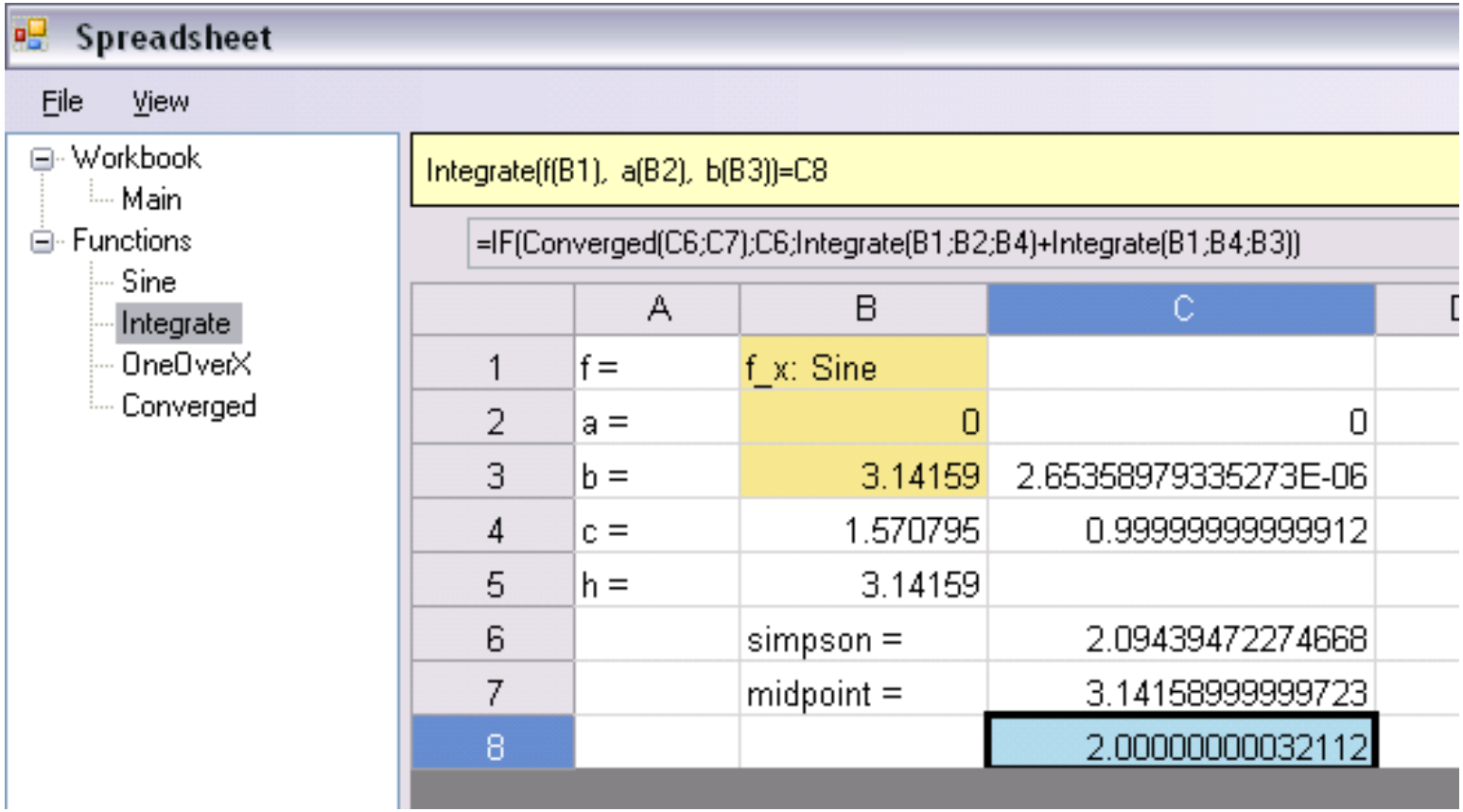
# Using the integrator

- The integral of sine from 0 to pi is 2
- That of 1/x from 1 to 54.6 is ln(54.6) = 4



# The Integrate sheet

- Compute integral two ways
- If results differ, subdivide recursively




| | A | B | C |
|---|---|---|---|
| 1 | f = | f_x: Sine | |
| 2 | a = | 0 | 0 |
| 3 | b = | 3.14159 | 2.65358979335273E-06 |
| 4 | c = | 1.570795 | 0.99999999999912 |
| 5 | h = | 3.14159 | |
| 6 | | simpson = | 2.09439472274668 |
| 7 | | midpoint = | 3.14158999999723 |
| 8 | | | 2.00000000032112 |

# Prototype implementation

- Based on CoreCalc
- No GUI for defining functions
- Import from Excel, with function sheets named like **`@TriArea`**
- Argument and result areas defined by named range **`TriArea_signature`**
- Matrix values can be passed in cells

# Features

- Cortes & Hansen's implementation supports:
  - Functions with variable number of arguments
  - Function with matrix arguments and matrix result
  - Functions with function arguments (higher-order)
  - Recursive functions
  - Prototype has no GUI but imports sheets from Excel
- Experience
  - Examples from the life insurance industry
  - Radical simplification of complex sheets
  - Seasoned spreadsheet users like the idea
  - Higher-order functions and matrices go well together

# ITU-TR-2006-91: *A Spreadsheet Core Implementation in C#*

Contents:

- What is a spreadsheet
- Some spreadsheet literature and resources
- The CoreCalc implementation
- Alternative designs
- The support graph
- Runtime code generation
- Sheet-defined functions
- Extensions and projects
- 233 spreadsheet patents and applications

http://www.itu.dk/people/sestoft/corecalc/