P-ADIC NUMBERS IN FORM

Coenraad Marinissen

c.marinissen@nikhef.nl

June 24, 2026

Amsterdam

OUTLINE

- Motivation
- p-adic numbers 101
- Implementation
  - Flint
  - Form
- A worked out example
- Development process
  - Can AI do Form development?
  - Discuss good practices
- Conclusion and outlook

WHY P-ADICS IN FORM?

- Many HEP calculations reduce to huge rational functions
  - ➡ Coefficients depend on (many) kinematical variables and ϵ

- Finite field methods have been popular for this [1406.4513, 1904.00009, 1905.08019,...]
  - ➡ Avoid expression swell
  - ➡ Reconstruct exact rational functions from samples

- Rational reconstruction may still be expensive [1812.04586, 2110.07541,...]
  - ➡ May need many samples and prime fields

- p-adics give finite-field methods with precision [2203.04269, 2312.03672]
  - ➡ Work modulo p^N, not just modulo p
  - ➡ Construct Laurent expansion directly:
    one evaluation can contain several expansion coefficient.
  - ➡ Example follows later in this talk.

P-ADIC NUMBERS

- The p-adic numbers, with p a prime, come from an alternative way of defining the distance between two rational numbers.

Real numbers: large absolute value means far from zero

→ 7^5 is large

p-adic numbers: divisibility by p means close to zero

→ 7^5 is small as a 7-adic number

P-ADIC NUMBERS

- The p-adic numbers, with p a prime, come from an alternative way of defining the distance between two rational numbers.

Real numbers: large absolute value means far from zero

→ 7^5 is large

p-adic numbers: divisibility by p means close to zero

→ 7^5 is small as a 7-adic number

- Valuation:

→ $\begin{cases} \text{Let } x \in \mathbb{Z} & \rightarrow \quad x = p^v u \text{ where } p \nmid u \\ \text{Let } x = \dfrac{a}{b} \in \mathbb{Q} & \rightarrow \quad v(x) = v(a) - v(b) \end{cases}$

called the unit

- Absolute value:

For $x \in \mathbb{Q}$: $\quad |x|_p = \begin{cases} p^{-v(x)} & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$

P-ADIC SERIES

- Any p-adic number can be uniquely expanded as a p-adic series:

$$x = \sum_{n=v}^{\infty} c_n p^n \qquad \text{with} \qquad c_n \in \{0,\ldots,p-1\}$$

- Such series are analogous to power series in a small parameters.

  ➡ Remember: large powers of p are "small" according to the p-adic absolute value $|\cdot|_p$

Example: ($p = 7$)

$$\frac{194}{7} = 7^{-1}\left(5 \cdot 7^0 + 6 \cdot 7^1 + 3 \cdot 7^2\right) = 36.5_7$$

$$\frac{1}{2} = 4 + 3 \cdot 7^1 + 3 \cdot 7^2 + 3 \cdot 7^3 + \mathcal{O}(7^4)$$

Verify by multiplying both sides by 2

- Middle ground between finite fields and floating-point numbers:

  ➡ First digits behaves like a finite field

P-ADIC IMPLEMENTATIONS

- Finite precision p-adics:

$$\frac{1}{2} = 4 + 3 \cdot 7^1 + 3 \cdot 7^2 + 3 \cdot 7^3 + \mathcal{O}(p^4)$$

➡ We know this p-adic number modulo p^4, i.e. modulo 2401

- p-adics on a computer:

1. Similar to real numbers: as a floating-point representation

$$\frac{194}{7} = 7^{-1}\left(5 \cdot 7^0 + 6 \cdot 7^1 + 3 \cdot 7^2\right) = 36.5_7$$

➡ May have infinitely many coefficients to the left.

2. Store coefficients of the series expansion

3. Represent the p-adic number by an integer that is p-adically close to it

$$4 + 3 \cdot 7^1 + 3 \cdot 7^2 + 3 \cdot 7^3 = 1201$$

➡ **This talk!**

FORM IMPLEMENTATION: USER SIDE

What do we need?

- Instruction to start the p-adic system

```
#StartPadic <prime>, <precision>
```

- Statement to convert from rationals to p-adics

```
ToPadic;
```

- Statement to convert back to the rationals

```
PadicToRat;
```

- Instruction to close the p-adic system

```
#EndPadic
```

FORM IMPLEMENTATION: USER SIDE

What do we need?

- Instruction to start the p-adic system

```
#StartPadic <prime>, <precision>
```

```
#StartFloat <precision>[,MZV=weight]
```

- Statement to convert from rationals to p-adics

```
ToPadic;
```

```
ToFloat;
```

- Statement to convert back to the rationals

```
PadicToRat;
```

```
ToRational;
```

- Instruction to close the p-adic system

```
#EndPadic
```

```
#EndFloat
```

FLINT INTEGRATION (I)

Data type: `padic_t`

- Represents p-adic numbers to precision N, stored in the form $x = p^v u$
- These can be accessed with the following macros:

Unit u:

```
fmpz *padic_unit(const padic_t op)
```

Valuation v:

```
slong padic_val(const padic_t op)
```

Precision N:

```
slong padic_prec(const padic_t op)
```

Can be used as both an *`lvalue`* and and *`rvalue`*

FLINT INTEGRATION (I)

Data type: `padic_t`

- Represents p-adic numbers to precision N, stored in the form $x = p^v u$
- These can be accessed with the following macros:

Unit u:
```
fmpz *padic_unit(const padic_t op)
```

Valuation v:
```
slong padic_val(const padic_t op)
```

Precision N:
```
slong padic_prec(const padic_t op)
```

Can be used as both an *lvalue* and and *rvalue*

Context: `padic_ctx_t`

- Contains data pertinent to p-adic computations so that it doesn't have to be stored with each element individually.

```
void padic_ctx_init(padic_ctxt_t ctx, const fmpz_t p, slong min, slong max, enum padic_print_mode mode)
```

prime number p

Precomputed powers p^e with:

$$min \leq e \leq max$$

print mode

FLINT INTEGRATION (II)

Memory management:

No need for `ctx`

```
void padic_init2(padic_t rop, slong N)
void padic_clear(padic_t rop)
void padic_reduce(padic_t rop, padic_ctx_t ctx)
```

→ Ensures $p \nmid u$ and reduces $u \mod p^N$

FLINT INTEGRATION (II)

Memory management:

No need for `ctx`

```
void padic_init2(padic_t rop, slong N)
void padic_clear(padic_t rop)
void padic_reduce(padic_t rop, padic_ctx_t ctx)
```

→ Ensures $p \nmid u$ and reduces $u \mod p^N$

Asignment and conversion:

```
void padic_set_si(padic_t rop, slong op, const padic_ctx_t ctx)
void padic_set_fmpq(padic_t rop, const fmpq_t op, const padic_ctx_t ctx)
void padic_get_fmpq(fmpq_t rop, const padic_t op, const padic_ctx_t ctx)
```

FLINT INTEGRATION (II)

Memory management:

No need for `ctx`

```
void padic_init2(padic_t rop, slong N)
void padic_clear(padic_t rop)
void padic_reduce(padic_t rop, padic_ctx_t ctx)
```

→ Ensures $p \nmid u$ and reduces $u \mod p^N$

Asignment and conversion:

```
void padic_set_si(padic_t rop, slong op, const padic_ctx_t ctx)
void padic_set_fmpq(padic_t rop, const fmpq_t op, const padic_ctx_t ctx)
void padic_get_fmpq(fmpq_t rop, const padic_t op, const padic_ctx_t ctx)
```

Arithmetic:

```
void padic_add(padic_t rop, const padic_t op1, const padic_t op2, const padic_ctx_t ctx)
void padic_mul(padic_t rop, const padic_t op1, const padic_t op2, const padic_ctx_t ctx)
void padic_div(padic_t rop, const padic_t op1, const padic_t op2, const padic_ctx_t ctx)
```

FORM IMPLEMENTATION: USER SIDE

```
 12 > /* #[ License : */ ⋯
 38   /*
 39 >    #[ Includes: padic.c ⋯
113 v    #[ Helpers :
114 >       #[ AllocatePadicPrintBuffer : ⋯
142 >       #[ InitPadicAux : ⋯
154 >       #[ ClearSinglePadicAux : ⋯
166 >       #[ AllocatePadicAux : ⋯
204 >       #[ ClearPadicAux : ⋯
231 >       #[ StartPadicSystem : ⋯
283 >       #[ ClearPadicSystem : ⋯
311      #] Helpers :
312 v    #[ Internal p-adic function format :
313 >       #[ Explanations : ⋯
329 >       #[ TestPadic : ⋯
396 >       #[ UnpackPadic : ⋯
468 >       #[ PackPadic : ⋯
574      #] Internal p-adic function format :
575 v    #[ Rekenen :
576 >       #[ FormRatToFmpq : ⋯
613 >       #[ MulRatToPadic : ⋯
637 >       #[ MulPadics : ⋯
659 >       #[ DivPadics : ⋯
689 >       #[ InvPadic : ⋯
717 >       #[ PadicReconstruct : ⋯
876      #] Rekenen :
877 v    #[ Printing :
878 >       #[ PrintPadic : ⋯
922      #] Printing :
923 v    #[ Compiler/runtime statements :
924 >       #[ CoToPadic : ⋯
947 >       #[ CoPadicToRat : ⋯
969 >       #[ CoFromPadic : ⋯
999 >       #[ ToPadic : ⋯
1047 >      #[ PadicToRat : ⋯
1117 >      #[ FromPadic : ⋯
1139     #] Compiler/runtime statements :
1140 v    #[ Sorting :
1141 >       #[ AddWithPadic : ⋯
1274 >       #[ MergeWithPadic : ⋯
1388     #] Sorting :
1389  */
```

Most of the code for the p-adics can be found in `padic.c`.

- We only need a few places in the rest of Form to connect to this
- This also contains all the code that connects to Flint

FORM IMPLEMENTATION: USER SIDE

What do we need?

- A data type for the p-adic number:

```
padic_(v,N,u);
```

➡ New function (**PADICFUN**) which has protected status from the pattern matcher etc.

➡ Arguments encode Flint's **padic_t** data type

```
312  #[ Internal p-adic function format :
313  >     #[ Explanations :…
329  >     #[ TestPadic :…
396  >     #[ UnpackPadic :…
468  >     #[ PackPadic :…
574  #] Internal p-adic function format :
```

- Routines to convert Flint's **padic_t** to Form's **float_** function and vice versa

```
405  static int UnpackPadic(padic_t out, WORD *fun)
```

```
476  static int PackPadic(WORD *fun, padic_t in)
```

- Routine to check that **padic_** is a valid p-adic number

```
338  int TestPadic(WORD *fun)
```

FORM IMPLEMENTATION: USER SIDE

What do we need?

- Code in **`Normalize`** to combine two or more **`padic_`** functions, or a **`padic_`** and a rational coefficient.

➡ Routines to do the p-adic arithmetic:

- Multiplication of two p-adics
- Division of two p-adics
- Inversion of a p-adic
- Multiplication of a p-adic with a rational

```
575 ⌄   #[ Rekenen :
576 ›       #[ FormRatToFmpq :…
613 ›       #[ MulRatToPadic :…
637 ›       #[ MulPadics :…
659 ›       #[ DivPadics :…
689 ›       #[ InvPadic :…
717 ›       #[ PadicReconstruct :…
876     #] Rekenen :
```

- Code in the sort routines that allows the addition of two terms.

```
1140 ⌄   #[ Sorting :
1141 ›       #[ AddWithPadic :…
1274 ›       #[ MergeWithPadic :…
1388     #] Sorting :
```

FORM IMPLEMENTATION: USER SIDE

What do we need?

- Routines to start and close the p-adic system:

```
#StartPadic <prime>, <precision>
```

```
#EndPadic
```

➡ Make the proper allocations etc.

```
113 ⌄   #[ Helpers :
114 ›       #[ AllocatePadicPrintBuffer :⋯
142 ›       #[ InitPadicAux :⋯
154 ›       #[ ClearSinglePadicAux :⋯
166 ›       #[ AllocatePadicAux :⋯
204 ›       #[ ClearPadicAux :⋯
231 ›       #[ StartPadicSystem :⋯
283 ›       #[ ClearPadicSystem :⋯
311     #] Helpers :
```

- Print routine

```
877 ⌄   #[ Printing :
878 ›       #[ PrintPadic :⋯
922     #] Printing :
```

- Compiler and runtime routines for the new statements

```
ToPadic;
```

```
PadicToRat;
```

```
FromPadic;
```

```
923 ⌄   #[ Compiler/runtime statements :
924 ›       #[ CoToPadic :⋯
947 ›       #[ CoPadicToRat :⋯
969 ›       #[ CoFromPadic :⋯
999 ›       #[ ToPadic :⋯
1047 ›      #[ PadicToRat :⋯
1117 ›      #[ FromPadic :⋯
1139    #] Compiler/runtime statements :
```

TOPADIC AND PADICTORAT

```
    #StartPadic 7,8
    Local F = 101/13;
    ToPadic;
    Print "<1> %t";
    PadicToRat;
    Print "<2> %t";
    .end

<1>  + (4 + 6*7^2 + 4*7^3 + 2*7^4 + 5*7^5 + 3*7^6)
<2>  + 101/13
```

- `PadicToRat` makes use of the code for `makerational_` for the rational reconstruction.
- `#StartPadic` checks if the given prime is an actual prime.

TOPADIC AND PADICTORAT

```
#StartPadic 7, -1
Local F = 101/13*7^-5;
ToPadic;
Print;

F =
   (4*7^-5 + 6*7^-3 + 4*7^-2);

.end
```

- `PadicToRat` makes use of the code for `makerational_` for the rational reconstruction.
- `#StartPadic` checks if the given prime is an actual prime.
- The precision in `#StartPadic` can also be zero or negative.

FROMPADIC

```
CFunction f;
#StartPadic 7,8
Local F = 101/13;
ToPadic;
FromPadic f;
Print;

F =
   f(0,8,443454);

.end
```

- `FromPadic` replaces the protected `padic_` function by a regular function.
  - ➡ The arguments of the function can be meaningfully changed.

FROMPADIC

```
CFunction f;
#StartPadic 7,8
Local F = 101/13;
ToPadic;
FromPadic f;
Print;
.sort

F =
   f(0,8,443454);


Symbol v,N,u;
id f(v?,N?,u?) = makerational_(u,7^(N-v))*7^v;
Print;
.end

F =
   101/13;
```

- `FromPadic` replaces the protected `padic_` function by a regular function.

  ➡ The arguments of the function can be meaningfully changed.

FROMPADIC

```
 CFunction f;
 #StartPadic 7,8
 Local F = 101/13;
 ToPadic;
 Print;
 .sort

F =
   (4 + 6*7^2 + 4*7^3 + 2*7^4 + 5*7^5 + 3*7^6);



 FromPadic f;
 Transform f decode(10,3):base=7;
 Print;
 .end

F =
   f(0,8,4,0,6,4,2,5,3,0);
```

- `FromPadic` replaces the protected `padic_` function by a regular function.

  → The arguments of the function can be meaningfully changed.

TEST CASE: FORCER

Potential bottleneck in Forcer: run out of disk space

- Too many terms
- Even in expansion mode, we could reduce the size of the coefficients

Solution 1: Finite fields

- Reconstructs full polynomial in ϵ
- Need many samples, even if we are only interested in a few orders in the ϵ-expansion.

Solution 2: p-adic numbers

- One p-adic number probes many coefficients in the ϵ-expansion
- Need more size compared to finite fields, but don't need many samples

Reconstructing Laurent expansion of rational functions using p-adic numbers:

$$R(\epsilon) = \frac{N(\epsilon)}{D(\epsilon)} = \sum_{k=k_0}^{\infty} c_k \epsilon^k$$

[2506.08452]

➡ Has the form of a p-adic expansion if we set $\epsilon = p$

TEST CASE: FORCER

Problem: we want to find the coefficients c_k in the Laurent expansion:

$$R(\epsilon) = \frac{N(\epsilon)}{D(\epsilon)} = \sum_{k=k_0}^{\infty} c_k \epsilon^k$$

1. Evaluate with $\epsilon = p$
2. We obtain a p-adic number $\quad R(p) = \sum_{k=k_0}^{\infty} b_k p^k = \sum_{k=k_0}^{\infty} c_k p^k$
3. But, cannot directly assume $b_k = c_k$

   ➡ b_k is a p-adic number itself: $\quad b_k = \sum_{j=0}^{\infty} b_{k,j} p^j$

4. We can extract the following relations:

$$c_k = b_{k,0}$$

$$c_{k+1} = b_{k,1} + b_{k+1,0} \mod p$$

$$c_{k+2} = \dots$$

TEST CASE: FORCER

Example: $R(\epsilon) = \dfrac{14\epsilon^2 + 12\epsilon - 3}{5\epsilon^2 + 8\epsilon} = -\dfrac{3}{8\epsilon} + \dfrac{111}{64} + \dfrac{341}{512}\epsilon + \dfrac{1705}{4096}\epsilon^2 + \mathcal{O}(\epsilon^3)$

```
#$prime = 1302397;

Symbol ep,x1,x2,v,N;
CFunction padic,rat,coeff;

#StartPadic `$prime',3
Local F = rat(14*ep^2 + 12*ep - 3,
              5*ep^2 + 8*ep);
Multiply replace_(ep,$prime);
id rat(x1?,x2?) = x1/x2;
ToPadic;
.sort

#do v=-1,2
    FromPadic, padic;
    id padic(?x1) = padic(?x1)+padic_(?x1);
    Transform decode(13,3):base=`$prime';
    id padic(`v',N?,x1?,?x2) = coeff(`v',x1)-makerational_(x1,$prime)*$prime^`v';
    id padic(?x1) = 0;
    .sort
#enddo
id coeff(v?,x1?) = makerational_(x1,$prime)*ep^v;
Print;

F =
   111/64 - 3/8*ep^-1 + 341/512*ep + 941/31*ep^2;

.end
```

TEST CASE: FORCER

And now for Forcer

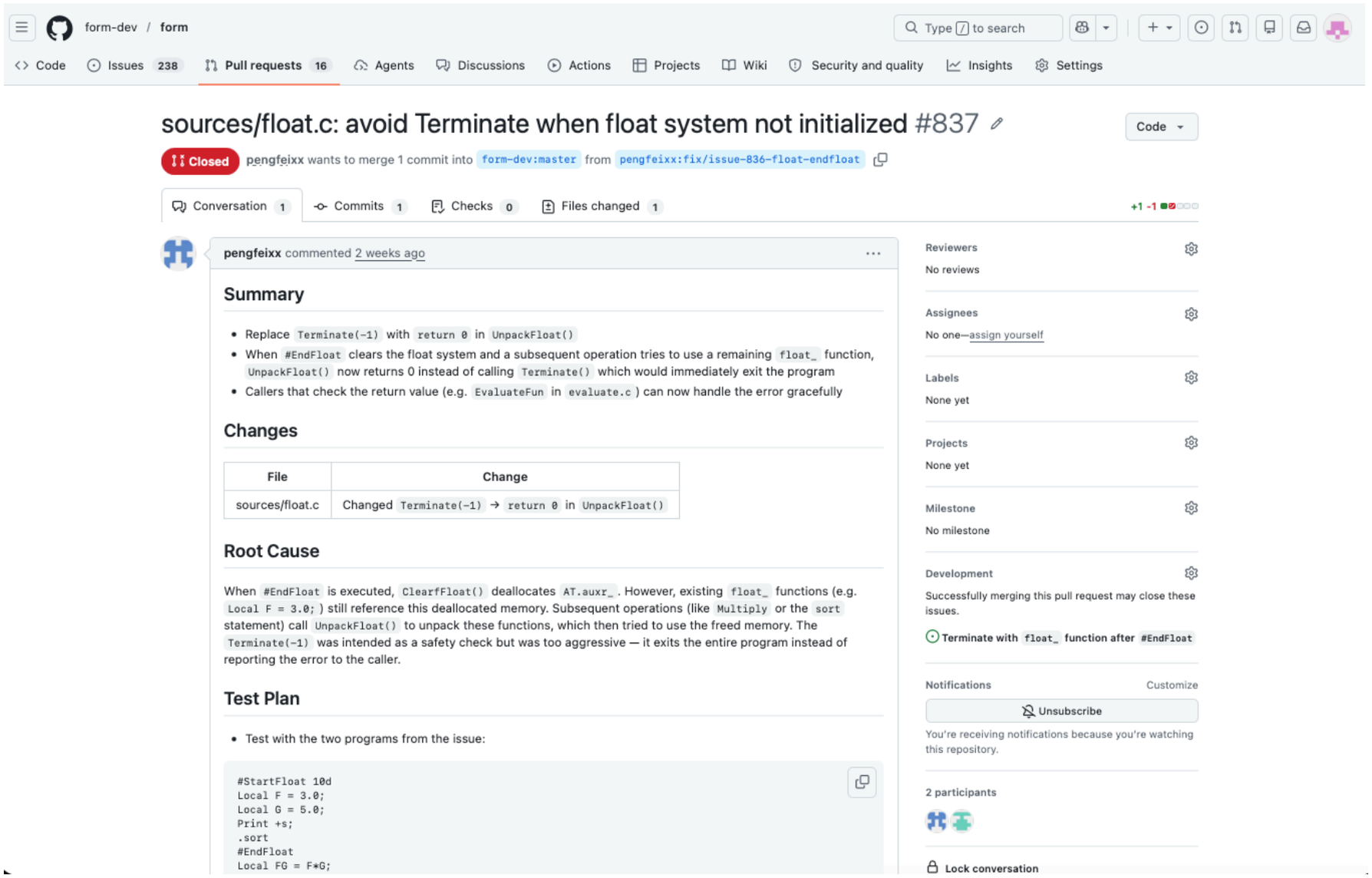
- Only need to change Forcer in a few places:
  - ➡ Remove the declarations of `ep`, `rat` and `RAT`
  - ➡ Disable `PolyRatFun` in a few places

```
Table ep();
Table rat(x?,y?);
Table RAT(x?,y?);
Fill ep = `PRIME';
Fill rat() = x/y;
Fill RAT() = y/x;
```

- Low complexity integrals:
  - ➡ Run time is approximately the same
  - ➡ Size is cut in half
- How does this scale to integrals of higher complexity integrals?
- Or a full physics calculation?

DEVELOPMENT PROCES

Can AI help with Form development?



Not this...

DEVELOPMENT PROCES

Can AI help with Form development?

- I already had a good understanding of the float implementation and it was clear that the p-adics could be implemented in a similar fashion.

→ ***Context***

→ Already knew Flint's implementation, so knew how `padic_` should be packed.

→ Easy to point to the float implementation:

```
float.c
```

```
evaluate.c
```

```
#ifdef WITHFLOAT
```

→ Had a good idea what statements should be added.

```
#StartPadic/#EndPadic
```

```
ToPadic;
```

```
PadicToRat;
```

→ Had a good idea what routines should be made for a working p-adic system

DEVELOPMENT PROCES

Used Codex with GPT-5.3 (February)

- Credits to Thomas Marinissen who let me use his pro account and helped me with good prompts.

- Supervised loop:

1 Analyze

- Fully understand and prepare
- Identify anything that is unclear: `questions.md`
- Do not implement yet!

2 Analyze answer

- First let Codex answer
- Then fill in the gaps myself

3 Implementation plan

- Make a plan
- Ask for small subtasks.
- Went back and forth until I was happy with the plan.

4 Implement

- Gave some last implementation requirements.
- Let it make tests such that it can correct itself.

DEVELOPMENT PROCES

Used Codex with GPT-5.3 (February)

- Credits to Thomas Marinissen who let me use his pro account and helped me with good prompts.

- Supervised loop:

1 Analyze
- Fully understand and prepare
- Identify anything that is unclear: `questions.md`
- Do not implement yet!

2 Analyze answer
- First let Codex answer
- Then fill in the gaps myself

3 Implementation plan
- Make a plan
- Ask for small subtasks.
- Went back and forth until I was happy with the plan.

4 Implement
- Gave some last implementation requirements.
- Let it make tests such that it can correct itself.

- First test worked immediately!
- A second test did not, but this was just a one-line fix.
- A few iterations later and it also had proper printing, and already code that was more tight and cleaned up.
- It all took less than a day → First implementation just 26 minutes.

DEVELOPMENT PROCESS

- Then began the reviewing/debugging...
  - ➡ Went over everything line by line.

Worked well

- Localized changes (of course it had a lot of context from the floats).
- Repetitive edits across many files.
- Reuse of existing macros.
- It can verify itself with small experiments and test programs.
- Good in hunting down bugs.
- Good in C syntax.

Worked less

- Unnecessary defensive checks.
- Excessive splitting into tiny helpers.
- Sometimes many iterations, e.g. took me ages before it understood Form's code folding.
- Did not always found helper functions that already exist in Form.
- Weak in writing Form scripts.

FORM DEVELOPMENT AND AI

Verdict

- Despite the long sessions reviewing and debugging Codex's output, I think I spend less time implementing the new functions.
- Codex is good in finding bugs in its own code.
- Trigger happy, so good to let it analyse first.
- It is good at small well defined tasks, not at vaguely-defined missions ("refactor all of Form").
- It helped to formulate a specific task where I have a decent idea in advance what the result ought to look like.
- Quite good at explaining the existing code → helpful for new developers

FORM DEVELOPMENT AND AI

More experiments:

- Fixing existing bugs (but with the proper context...)
- Discovering new bugs
- Refactoring
- Optimising code
- How good is it in math? Making algorithms from lemmas and theorems?

Do we need a specific AI policy for Form?

- Don't want a lot of AI slop.
- Mention model etc. in PR message.
- Is reviewing AI-written code really different from reviewing human-written code?
- Keep the contributor responsible for understand and defending the change.
- Should AI-generated code require a different review process, or simply the same quality bar with explicit disclosure?

CONCLUSION AND OUTLOOK

- See PR #845 for the current status
- Open questions
  - What should happen if the user closes the p-adic system and opens it again with a different prime?
  - Should the prime number be saved in `padic_(v,N,u)`?
  - Have a `FromPadic f, series;`? Or update `Transform decode` to handle bigger primes?
- Outlook
  - Version 5.1
  - Broaden tests and documentation
  - Benchmark more real reconstructions
  - New Flint features for the p-adics coming soon: Flint #2719
  - Explore what AI can do for us. Good discussion point for this week as well.