

CLAUDE CODE にブラックホールを 衝突させたらAI駆動開発で 大事なものが見えてきた (かもしれない)



自己紹介

- さめ (meΓ-cck)
 -  VRChat 物理学集会の主催
 -  社会人学生として通信制大学
在学中
- 得意分野:
 -  コンピュータビジョン (画像
認識 / 点群処理)
 -  空間情報処理 (地理情報 / リ
モートセンシング)
 -  クラウドインフラ設計 / IaC
(AWS, GCP)
- 学生時代は地球物理学を専攻
- 地球観測技術のエンジニアとして活
動中



今日話すこと

- 連星ブラックホール(BBH)合体の数値シミュレーションをAIエージェント(Claude Code)と一緒にやった
- どんなふうの開発にAIエージェントを使ったか？
- AIエージェントを利用した開発で得た教訓
- 一般のソフトウェアやITサービス開発にも役立つ知見



今日話さないこと

- BBH 合体の数値シミュレーションの詳細
 - 本題に必要な話題だけざっくり話します
 - 興味がある方は物理学集会での発表スライドを後で共有します！
- Claude CodeをはじめとしたAIエージェントの使い方の一般論
- Claude Codeのさまざまな設定

何をやった？

BBH合体(GW150914)シミュレーション
の概要

そもそものモチベーション

- ゴールデンウィークに旅行に行く予定があった
- 2日半パソコンもインターネットも使えない場所で過ごす必要
- 「その間、家でパソコンに重い計算をやっておいてもらおう！」
 - 自分の集会でのネタにもなるし、ブラックホールのシミュレーションをやるうと思った

BBH合体はどんな現象？

- 2つのBHがグルグル回りながら合体して1つのBHになる
- 合体する時莫大なエネルギーが**重力波**として放出される
- 重力波は1916年にアインシュタインが理論的に予言
- 2015年に初観測 (GW150914)
 - 理論から実証まで約100年
 - 太陽質量3個分のエネルギーが重力波として放出
- 数値シミュレーションが実証に大きく貢献
 - 重力波の波源としてのBBHの理解を深めた
- YouTubeにわかりやすい動画があるので見てみましょう！

BBH合体シミュレーションの困難と 歴史

- 重力を記述するアインシュタイン方程式を解く必要がある
- アインシュタイン方程式は一言で言うともものすごく難しい
- 2005年に初めてシミュレーションに成功
- 1990年代に京都大学の柴田大先生、中村卓史先生など日本の物理学者が数値相対論の発展に大きく貢献！
 - 数値相対論: 重い天体の運動のような相対論効果が現れる現象を数値シミュレーションする分野

シミュレーションに利用したOSS

- Einstein Toolkit
 - 数値相対論のOSSフレームワーク
 - 公式ギャラリーにさまざまな現象の計算例が紹介
 - もともとスーパーコンピュータで動かすような計算をするので環境構築が地獄

**einstein**
toolkit

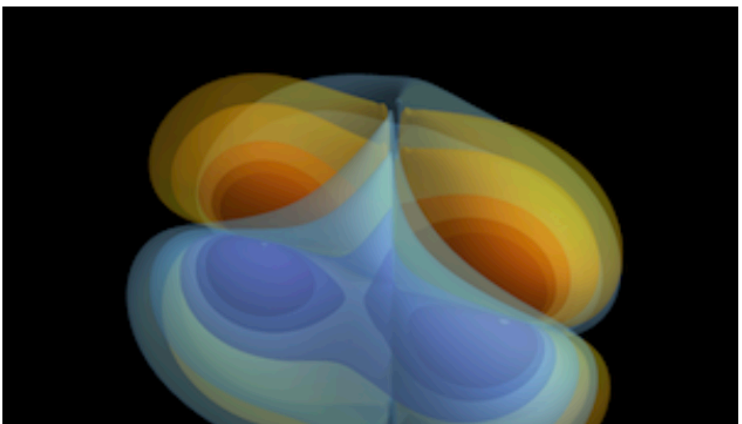
Home About Download Documentation Help! Contribute Gallery

CI failing

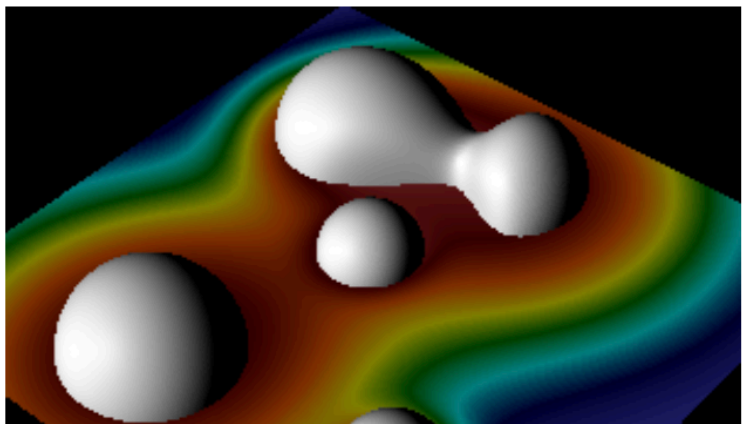
Einstein Toolkit Gallery

This page contains example simulations that can be run using the Einstein Toolkit, either exclusively or in combination with external codes. The parameter files and thornlists required to reproduce the simulations are provided. Some examples also include images and movies, analysis and visualisation scripts, example simulation data, and tutorials.

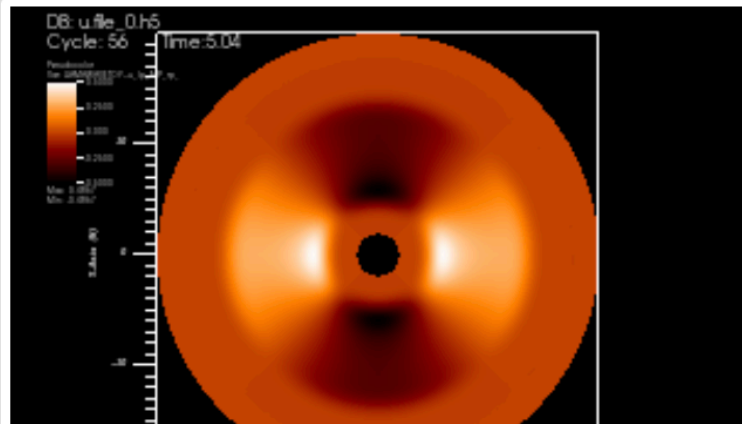
Binary black hole GW150914



Poisson equation



Multi Patch Scalar Wave Equation



GW150914のパラメータ

- Zenodo(データやシミュレーション結果を公開できるプラットフォーム)にGW150914のパラメータとシミュレーション結果を公開してくれている人がいた
 - [10.5281/zenodo.155394](https://doi.org/10.5281/zenodo.155394)
- Einstein Toolkitの開発環境を整えてパラメータをダウンロードすれば誰でもシミュレーションが実行可能
 - あくまで「原理的には」

AI エージェント (CLAUDE OPUS 4.7) を使った開発プロ セス

事例も交えて紹介

タスクの設計

- 最近のAIエージェントは**長時間、自律的に**依頼したタスクに取り組んでくれる
- コンテキストも長く保持できる
- 開発の粒度を小さくして頻繁にセッションを区切る必要はない
- **タスクと完了条件、「やらないこと」の設定が重要**
 - チーム開発で求められるスキルが活きる気がする (マネジメントとガバナンス)

タスクの設計例 (簡略版)

タスク

- DockerでEinstein Toolkit の開発環境をビルドする

完了条件

- Docker イメージのビルドの完了
- チュートリアルシミュレーションのドライランをパスする

やらないこと

- チュートリアルシミュレーション本番実行

- ゴールを明確かつ適切に定めれば、あとはAIが自律的にやってくれる
 - 実際はもう少し詳しく指示が必要、上記の例はわかりやすさ重視のイメージ

タスク設計の判断

- (依存関係の複雑さで悪名高き) Einstein toolkit をDockerでビルド
 - 環境分離に加えて、環境構築の手順をコードで残せる
 - Gitで管理できる
 - LLMと相性抜群
 - ビルド中にエラーが出ればエラーメッセージを元にDockerfile等を修正
 - 今回はDockerを利用したけど、MakefileやShell scriptでも同様のことが可能
 - 作業の標準化、再現性の担保はAI開発では重要
 - 何をやったかコードやコミットログに全部残せる

完了条件の設定

- Einstein Toolkit のチュートリアルを把握
 - 質量が等しい自転のない2つのBHが正面衝突するシミュレーション
 - 本番のシチュエーションと比べてシンプル
 - ただしそれでも完走には時間がかかる
 - ドライランのパスを完了条件に定めた
-
- 全体のゴールから適切な粒度のマイルストーンを定める
 - 自律的にいるんなことをやってくれるようになった分、やりすぎや脱線を防ぐために中間成果物の管理やQA、スコープ外タスクの定義が重要

各段階のマイルストーン

- チュートリアル完走
- GW150914のシミュレーションのドライラン
- シミュレーションを最初の数ステップのみ実行
 - 完走まで16日以上かかると判明、解像度を下げて3日で終わる設定に変更 (旅行中に終わらせるという制約から逆算、目標を定義し直す)
 - 中間成果物のレビューと「やらないこと」の提示が効いた局面

各段階のマイルストーン

- GW150914を全体の1/17 (100M) まで計算
 - Zenodoで公開されている結果と比較、解像度の変更の影響を検証
- 1000Mまでシミュレーションを実行
 - 合体に到達する時間が少し遅れるが回転の軌道は再現できることを確認
- 1700Mまで完走

※ Mは秒や分みたいな時間の単位だと思ってください

マイルストーンがなぜ大事か？

- 中間成果物の評価
 - 大きな手戻りを防げる
 - 方針転換が可能
 - 途中で異常があっても気付かず最後まで走らせると、
失敗箇所の特定が困難で計算リソースの浪費が大きい
- 進捗管理の可視化
 - GitHub Issues に各ステップの目標と小タスクを記述
 - GitHub CLIでAIエージェント自身にタスク設計と管理を任せられる
 - タスク完了でIssueをCloseすると単純に気持ちいい

各ステップでのユニットテスト

- ユニットテストは中間成果物を評価 (各ステップ終了時にテスト実行を指示)
- 今回は各段階でリファレンスとの差分を評価
- 通常のソフトウェア開発にも通じるはず

Check	Self-run	Reference	差分	閾値	Pass
マージャー時刻	925.1 M	898.7 M	+2.94%	±5%	✓
最終 BH 質量 M_f	0.9518	0.9527	-0.10%	±2%	✓
最終 BH スピン χ_f	0.6930	0.6877	+0.0054 abs	±0.02	✓
ψ_4 ピーク振幅	7.21e-4	7.34e-4	-1.79%	±10%	✓

マイルストーンをどう設定・運用するか？

- 自縄自縛かもしれないが、AIエージェントと相談しながら決めていい！
- IssueやWikiなどGitHubの機能を最大限に利用する
 - 開発者にもAIにも有用なドキュメントになる
 - GitHub CLIでAIエージェントに作成を依頼してもいい！
- 全体の見通しを立てることで想定外の事態が起こっても軌道修正しやすい
 - 今回の場合はそのままシミュレーションを実行すると16日かかるのが判明したこと

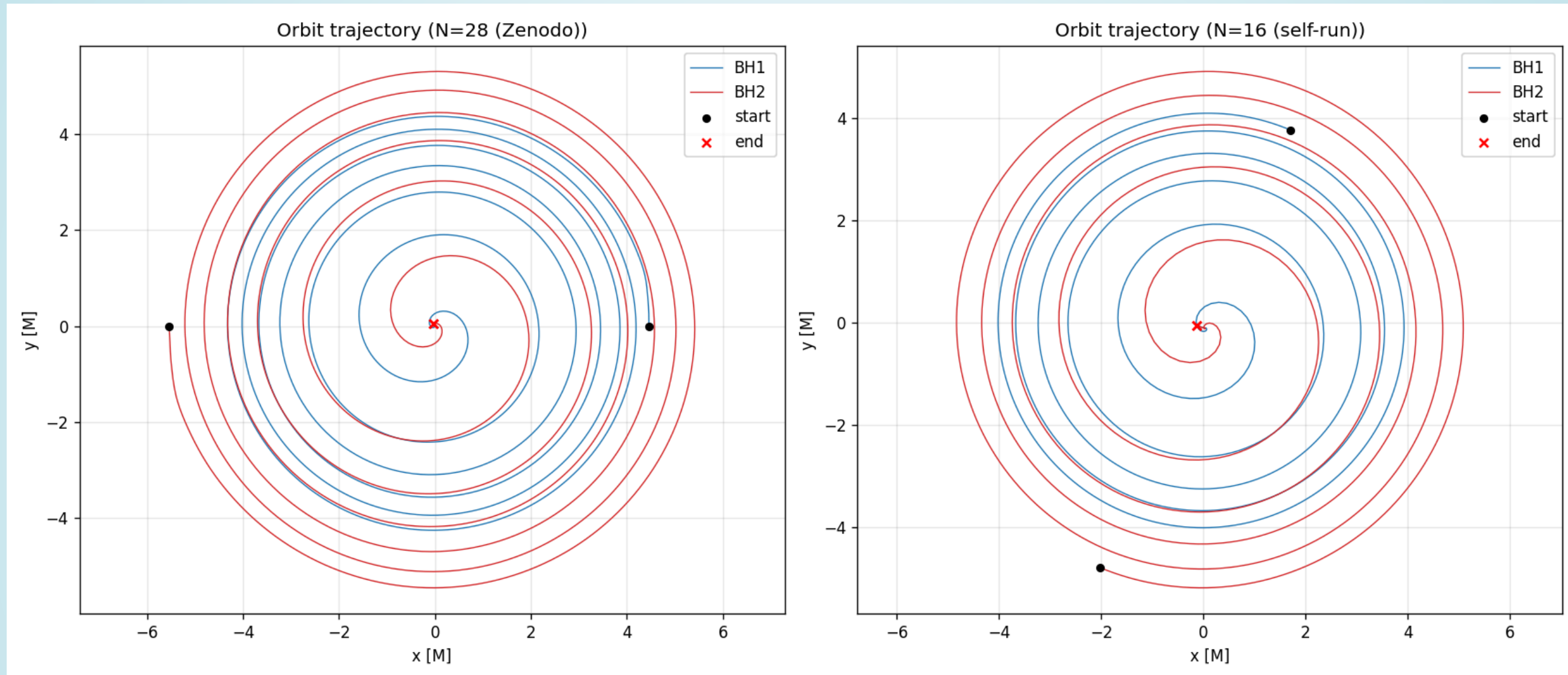
王道を征く

- 「ソフトウェア開発の基本しか言ってるじゃねーかコノヤロウ！」
- 「こんなんじゃLTにならないよ」
- 「そんなんじゃ甘いよ」
- そう思った人は正しい

- AIを使っても王道は変わらない
- **王道を征く**

シミュレーション結果

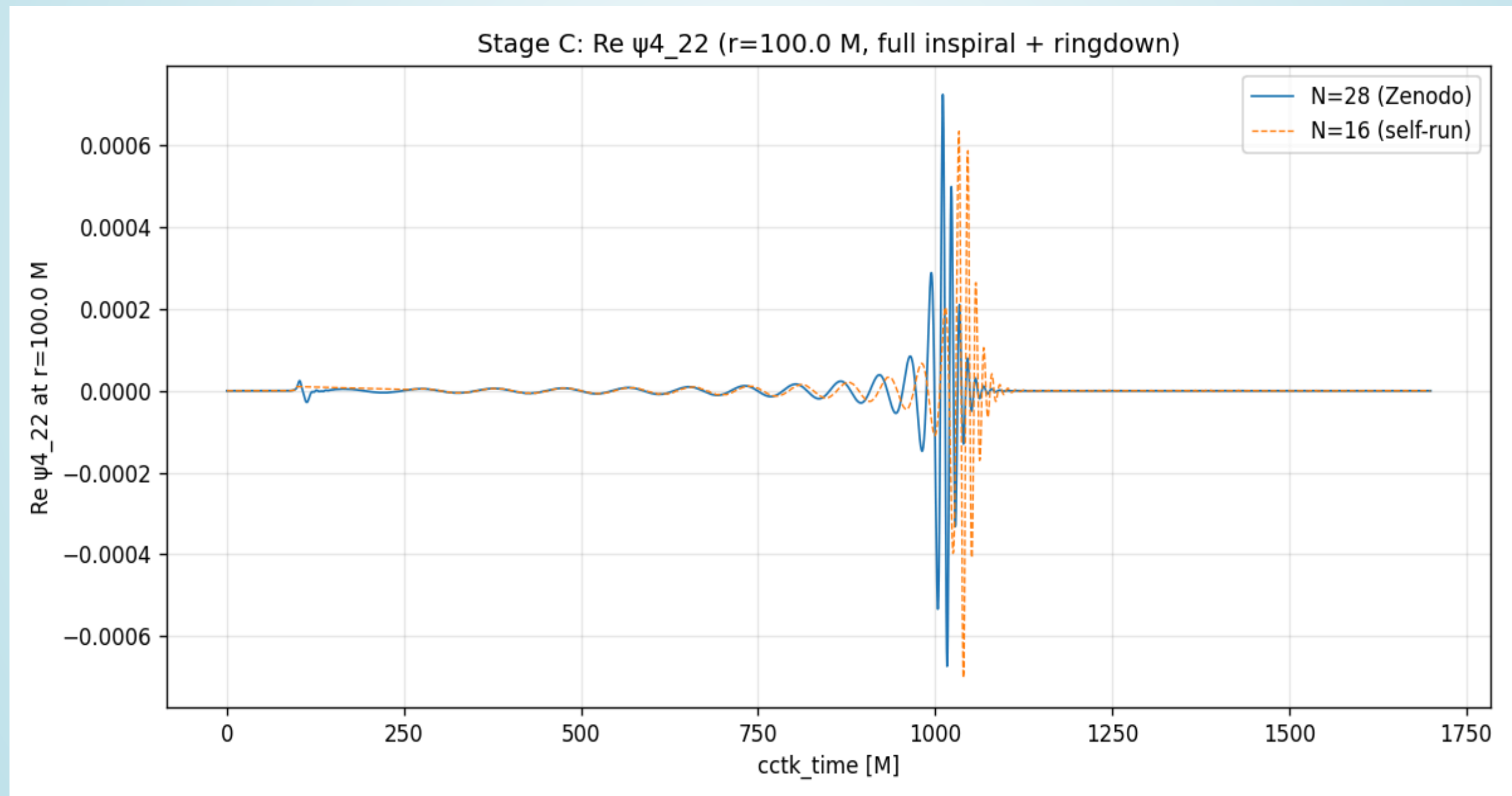
連星 BH の軌道 (XY 平面)



N=16 (本研究) と N=28 reference でほぼ重なる螺旋軌道。中心に向かって巻き込み、最後に合体

初期位置が違うのは出力結果の管理ミスです...

重力波 ψ_4 実部 — 位相は少しずつれる が形は再現



質量損失の計算

$$\Delta M = M - 0.9518M = 0.0482M \simeq 3.13M_{\odot}$$

M : 初期状態での2つのBHの総質量

- リファレンスの質量損失は約 $3.07M_{\odot}$ 。
- 「太陽質量の約3倍のエネルギーが重力波として宇宙に放出された」という有名な事実を数値シミュレーションで再現できた！

完走した感想

- インスパイラル期の軌道は極めて高い精度で再現できた
 - マージャー時刻が26M遅れるだけ
 - 重力波のピーク振幅は 1.79% しかずれない
 - 観測結果との定量的な検証には**明確に不足**だが、現象の定性的理解には十分
 - 質量損失は 3.13 太陽質量
-
- 太陽質量3個分のエネルギーが重力波として放出された、という有名な事実を再現できたことが嬉しかった
 - 歴史的に重要なイベントを自分のPCで走らせられたのは単純に嬉しい！

AI時代の開発の教訓

先人の言葉から学び、考える

自分がやったことを正確に評価する

- Einstein Toolkitのコードを書いたわけではない
- 数値相対論の定式化を理解したわけでもない
- 公式ギャラリー・LIGO 論文・後続の数値相対論論文で
やり尽くされている成果を自宅のPCで実行可能な条件に
ダウンサイズして実行しただけ
- **動かせることとなぜ、どのように動くか理解しているこ
とは完全に別物**

- 学術的新規性はない
- でも数値相対論からHPCの勉強ができた
- 歴史的なイベントを個人でも再現できることの喜び

柴田大先生の言葉から学び考える

- 柴田 大, 日本の数値相対論 (2010)より引用
 - 「だれでも出来る数値相対論の時代」
 - 「相対論を知らなくてもコードが作れる！」
 - 「公開も促進される：どこかで拾ってもよい！」
 - 「一方、つまらない/あやしい仕事も多数登場？困った輩も出るだろう(間違った結果を平然と出す輩が増える?)」
- 柴田先生は数値相対論の定式化に多大な貢献
- 「誰でも数値相対論ができる」ように基盤を整備した人

「誰でもできる」 @2010年

- 2010年の「誰でもできる」
 - 数値相対論の定式化を一からできなくとも
 - 数値シミュレーションやHPCの知識があり
 - リッチな計算資源にアクセスでき
 - 高額な科学計算の商用ソフトウェアである
Mathematicaを使えば
- 誰でもできる (正直かなり強い条件)

「誰でもできる」 @2026年

- 2026年の「誰でもできる」
 - AIエージェントに任せて
 - Einstein Toolkitをビルドし
 - 先行研究のシミュレーション結果を利用すれば
 - 個人所有のPCで
- 誰でもできる (2010年に比べて圧倒的に障壁が低い)
 - 実際に数値相対論を何も知らないわたしでも計算する
だけならできてしまう
 - 計算資源もクラウドを使うという選択肢がある

数値計算と物理

- 「数値計算はやった後に、物理(本質)を引き出す仕事が必要である。」
- 「3行で説明できないような計算結果は安易に信じては行けない。」
- 「計算結果を得た場合、3行で説明できるように熟考し、物理的説明を加えて論文にせよ。」

先述の柴田先生のスライドより引用

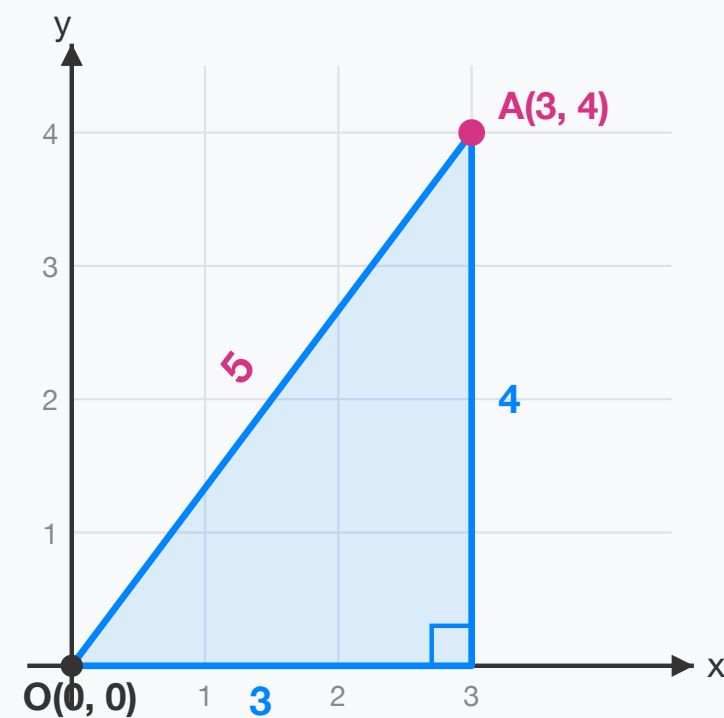
- **柴田先生の言葉はますます重要性を増しているのでは？**

個人の所感

AI時代にどう生きるか？

「なぜ？」を問い続ける

- 簡単な思考実験: 原点 $O(0, 0)$ と点 $A(3, 4)$ の距離を計算してください
- LLMにそれを計算するコードを書かせたらすぐに終わる
- でも「なぜそのような計算をするのか？」は答えられるようにすべき



```
import numpy as np

def calculate_norm(pnt1, pnt2):
    return np.linalg.norm(np.array(pnt1)
                           - np.array(pnt2)
                           )

def main():
    pnt1, pnt2 = [0, 0], [3, 4]
    ans = calculate_norm(pnt1, pnt2)
    print(ans)
```

「どのように」を答えられるように する

- 「直角三角形の辺の長さを考える。斜辺以外の辺の長さが3,4なので、2つの点の距離は斜辺の長さになり、三平方の定理から両点の距離は $\sqrt{3^2 + 4^2} = 5$ 」
- LLMがコードを書いてくれる。numpyやeigenなどの便利なライブラリの関数がある
- それらを使うなというわけではない
- でも何をやっているかは説明できるべき

実務での「なぜ？」の問いかけ

- 「なぜDBトランザクションにORMを使うのか？生のSQLではダメなのか？」
 - ORMはコードの可読性やサニタイズの観点で優位
 - 一方、生のSQLはパフォーマンスに優位点がある
 - 扱っているDBのレコード数や想定されるクエリの複雑さ、セキュリティや保守性を天秤にかけ、ORMを利用するメリットが上回ると判断した
- **LLMがORMを使う実装をした、ではNG！**

- **思想を語る**

自分がやりたいことを探し続ける

- 「やること」を決めたらそこから先の作業はAIがかなりの部分をやってくれる
- でも「何をやるか？」は自分で決め続けなければならない
 - 旅行をするから時間がかかる計算をしよう、というモチベーションを持つ
 - 重い計算の中でも人類初の重力波検出イベントを追試しよう、というセンス (他にも重い計算はいっぱいあるのに！)
 - 自分が何をやりたいか、何に興味を持てるか、という内なる声に耳を傾ける

世間の人は何を求めているのか考え続ける

- 「XXを集計して」「YYを計算して」と依頼すればLLMはすぐにやってくれる
- でも何をなぜ計算するのは人間が考えて決める
- 例えばあなたに仕事を依頼しているクライアントや、開発中のプロダクトの潜在的なユーザー
 - 彼らが何を求めているか？
 - どのような集計や機能実装をすれば役に立つか？
 - そのためにはどんなデータが必要で、どう評価すべきか？
- 根源的な課題や欲求を汗をかいて考え続ける

まとめ(平凡だけど多分大事)

- AIはBHの合体のような複雑な計算(2005年まで誰も計算できていなかった)をできるくらいに進歩している
- でも「何を」「なぜ」「どのように」機械にさせるのかは人間が考えて決める必要がある
- **AIが手を動かしてくれるからこそ、人間は汗をかいて泥臭く地道に考え続けるしかない！**
- AIが進歩し続けて「いるのに」、ではなく、「からこそ」たくさん考える、考えまくる
- マイルストーンも、思想を語ることも、考え続けることも全部「王道」。王道は変わらない。むしろ重みが増す
- **王道を征く**

APPENDIX

APPENDIX: SIBLING PROJECTS

- VRChat物理学集会 LT (GW150914 シミュレーションの報告): <https://github.com/s-sasaki-earthsea-wizard/vrc-gw150914-einstein-toolkit-LT>
- 元プロジェクト: <https://github.com/s-sasaki-earthsea-wizard/gw150914-einstein-toolkit>