並列計算

現在のプログラム（逐次プログラム）

- 初期値設定

  全ての恒星の初期位置と速度を設定

■ 以下を繰り返す

  - for(i=0 ; i < n ; i++)

    星iが全ての星から受ける力の和を計算

  - for(i=0 ; i < n ; i++)

    星iの速度と位置の変化を計算

並列化

- 並列化
  - 一つの問題を分割し複数のプログラム（以後プロセス）で同時に計算
  - 分割方法
    - 領域分割、機能分割、両者の組み合わせ

  →ここでは領域分割法を利用

領域分割　例:プロセス4個、星100個

プロセス0
- 初期値設定

■以下を繰り返す
- for(i=0 ; i < 25 ; i++)
  星iが**全ての星**から受ける力の和を計算★
- for(i=0 ; i < 25 ; i++)
  星iの速度と位置の変化を計算

プロセス1
- 初期値設定

■以下を繰り返す
- for(i=25 ; i < 50 ; i++)
  星iが**全ての星**から受ける力の和を計算★
- for(i=25 ; i < 50 ; i++)
  星iの速度と位置の変化を計算

プロセス2
- 初期値設定

■以下を繰り返す
- for(i=50 ; i < 75 ; i++)
  星iが**全ての星**から受ける力の和を計算★
- for(i=50 ; i < 75 ; i++)
  星iの速度と位置の変化を計算

プロセス3
- 初期値設定

■以下を繰り返す
- for(i=75; i < 100 ; i++)
  星iが**全ての星**から受ける力の和を計算★
- for(i=75 ; i < 100 ; i++)
  星iの速度と位置の変化を計算

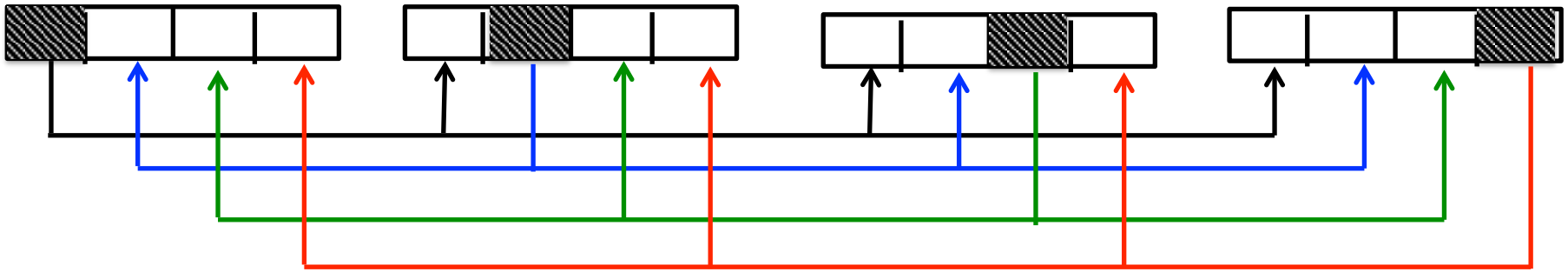
注:これだけでは動かない
★の位置では、全ての星の最新の位置データが必要

データ交換　ブロードキャスト

毎回ループの終わりで、計算した星の位置を他の全てのプロセスに送信

プロセス0	プロセス1	プロセス2	プロセス3
•初期値設定	•初期値設定	•初期値設定	•初期値設定
■以下を繰り返す	■以下を繰り返す	■以下を繰り返す	■以下を繰り返す
•for(i=0 ; i < 25 ; i++)	•for(i=25; i<50;i++)	•for(i=50;i < 75 ; i++)	•for(i=75;i <100;i++)
……	……	……	……
•for(i=0 ; i < 25 ; i++)	•for(i=25; i<50;i++)	•for(i=50;i < 75 ; i++)	•for(i=75;i <100;i++)
……	……	……	……
ブロードキャスト	ブロードキャスト	ブロードキャスト	ブロードキャスト

斜線部は各プロセスが
担当するデータの場所



SPMDモデル

- SPMD(single program multiple data)

  プログラムは一種類で、各プロセスで計算するデータのみが異なるモデル

- プログラムの統一

  計算範囲を式で与える

プロセス0	プロセス1	プロセス2	プロセス3
•初期値設定	•初期値設定	•初期値設定	•初期値設定
nn=n/p	nn=n/p	nn=n/p	nn=n/p
Is=nn*rank	Is=nn*rank	Is=nn*rank	Is=nn*rank
Ie=is+nn	Ie=is+nn	Ie=is+nn	Ie=is+nn
■以下を繰り返す	■以下を繰り返す	■以下を繰り返す	■以下を繰り返す
•for(i=is ; i < ie ; i++)	•for(i=is; i<ie;i++)	•for(i=is;i < ie ; i++)	•for(i=is;i <ie;i++)
……	……	……	……
•for(i=is; i < ie ; i++)	•for(i=is; i<ie;i++)	•for(i=is;i < ie ; i++)	•for(i=is;i <ie;i++)
……	……	……	……
ブロードキャスト	ブロードキャスト	ブロードキャスト	ブロードキャスト

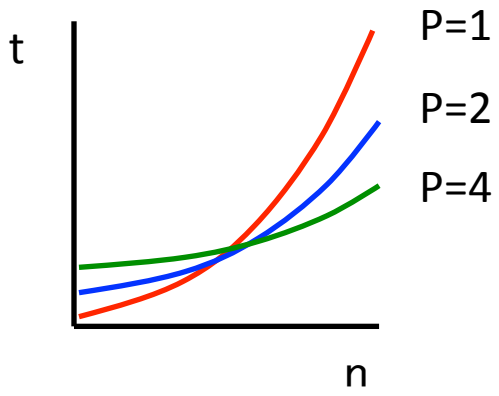
p:プロセス数、 rank: プロセス番号, n: 星の総数

MPI

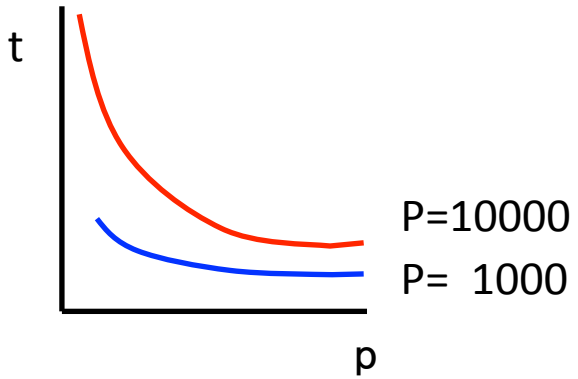
- 今回の問題の並列化に必要な操作
  - プロセス生成
  - プロセス数とプロセス番号の取得
  - ブロードキャスト
- MPI
  - メッセージパッシングを用いた並列処理のためのライブラリ
  - SPMDモデル
  - MPIを用いて書いたプログラムは、MPIが実装されていれば他の並列計算機やPCクラスタ等でも利用可能

処理時間の測定

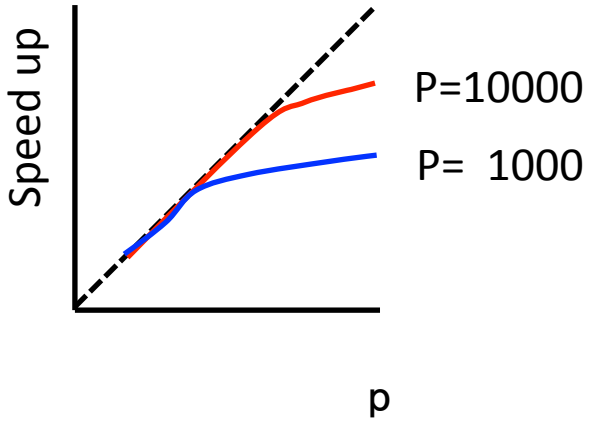
並列化による計算時間の変化1



並列化による計算時間の変化2



プロセス数による高速化率の変化



処理時間の測定

- 処理時間

  →実行に必要な実時間

- 処理時間の見積もり
  - 処理時間＝計算時間(t_{para})＋通信時間(t_{comm})
  - 計算時間：並列化によって減少 $t_{para}=t_{single}/p$
  - 通信時間：単純ではない

通信時間の見積もり

- ブロードキャスト送信されるデータ量
  - data=n*8*2/p*(p-1)*p byte
  - t_{comm}=data*8/通信速度 (1Gb/s ←理想値)
    - パケット化等により通信速度低下
    - t_{comm}=data*8/実効通信速度＋C*通信回数

    （実効通信速度とCは、通信の実験をして求める）
  - ブロードキャストの通信アルゴリズムによっては、通信時間が減少

    木構造の場合

    t_{comm}=n*8*2/p*log(p)*p*8/実効通信速度

    +c＊通信回数

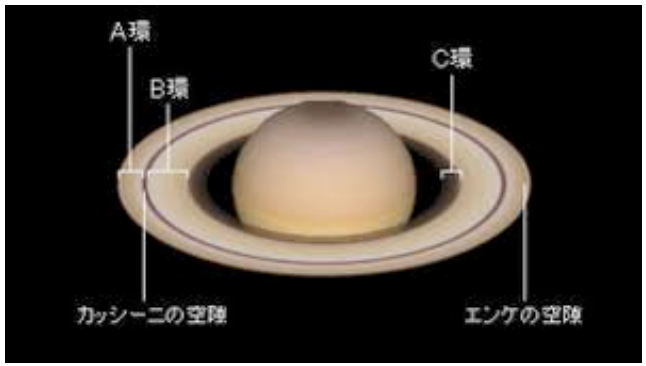
選択課題の補足

小惑星帯の形成

一方が非常に**重く**、かつ二つの惑星の**周期が整数比**の場合
相互作用（共振効果）により、軽い方の惑星が軌道から飛び出す
例：カッシーニの隙間、カークウッドの隙間等

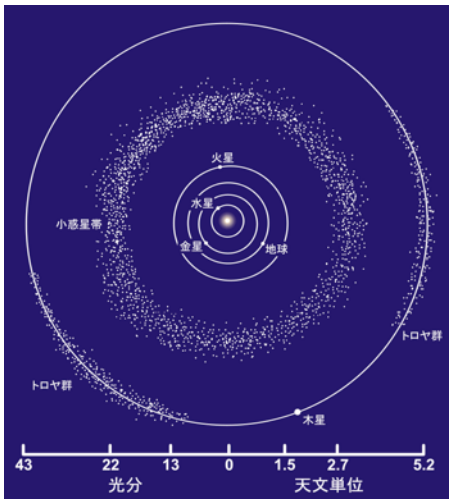
周期 T と円軌道の半径の関係
→ $T^2=(4\pi^2/(GM)) * r^3$ (4.13)

土星の輪におけるカッシーニの隙間

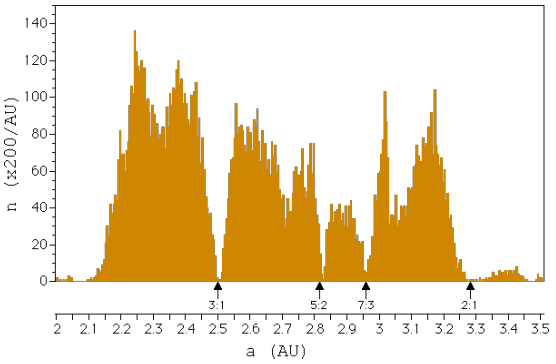


https://www.google.com/search?q=%E3%82%AB%E3%83%83%E3%82%B7%E3%83%BC%E3%83%8B%E3%81%AE%E9%9A%99%E9%96%93&client=firefox-b-d&tbm=isch&source=iu&ictx=1&fir=XXiN3Hg_q5wCBM%253A%252C4ffc5pOWAQSugM%252C_&vet=1&usg=AI4_-kRBjV05mR7150wGrp6DtPTx53l7hQ&sa=X&ved=2ahUKEwi3u5DXt5HkAhVCJaYKHZ8pDB8Q9QEwA3oECAkQBg#imgrc=qPLrm0kURBZPaM:&vet=1

小惑星帯におけるカークウッドの隙間



https://ja.wikipedia.org/wiki/%E5%B0%8F%E6%83%91%E6%98%9F#/media/%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB:Asteroid_Belt_ja.svg

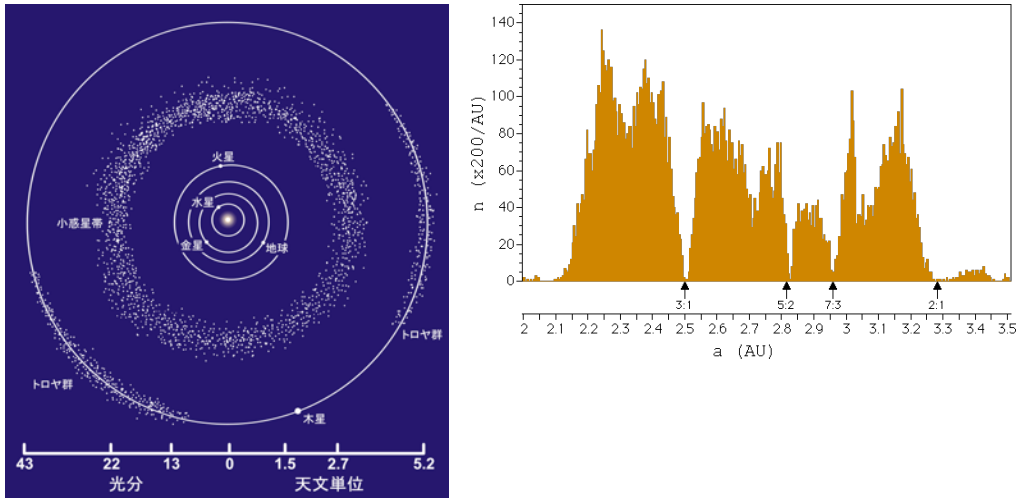


https://upload.wikimedia.org/wikipedia/commons/6/64/Kirkwood_Gaps.png

銀河の形成　#1

- 質量が重い恒星が一つある場合の、恒星の軌道分布（小惑星帯の銀河版）

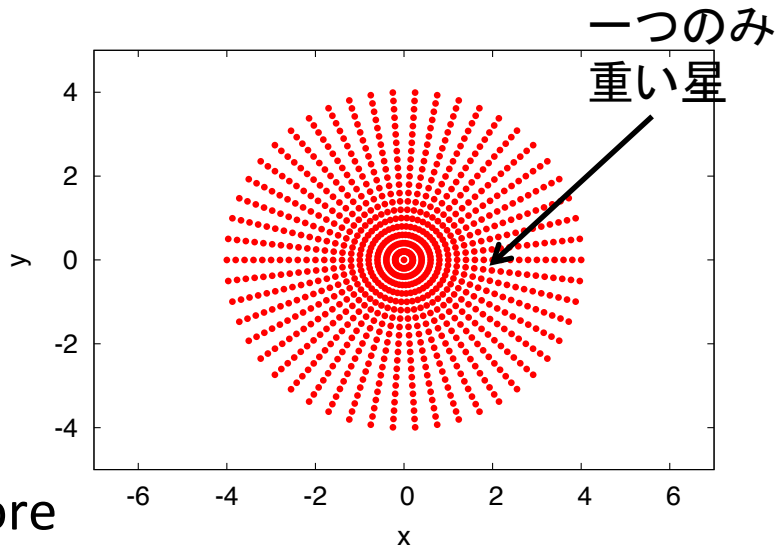
小惑星帯におけるカークウッドの隙間



- 期待すること
  - 恒星の半径に対する頻度分布に偏りが生じる
  - 偏りは重い恒星の周期との関係で説明できる

銀河の形成　#2

- やること
  - できるだけ多くの星を配置
  - 一つの星だけ質量増加
    - 例：他の星の300倍
  - 長時間計算
  - 半径に対する頻度分布作成
- 計算例
  - 星の数1000個
  - 半径4の範囲で分布
  - 重い星の初期位置(2,0)
  - シミュレーション時間　400
  - 計算時間
    - 半日程度by Xeon (2.6GHz) 1 core

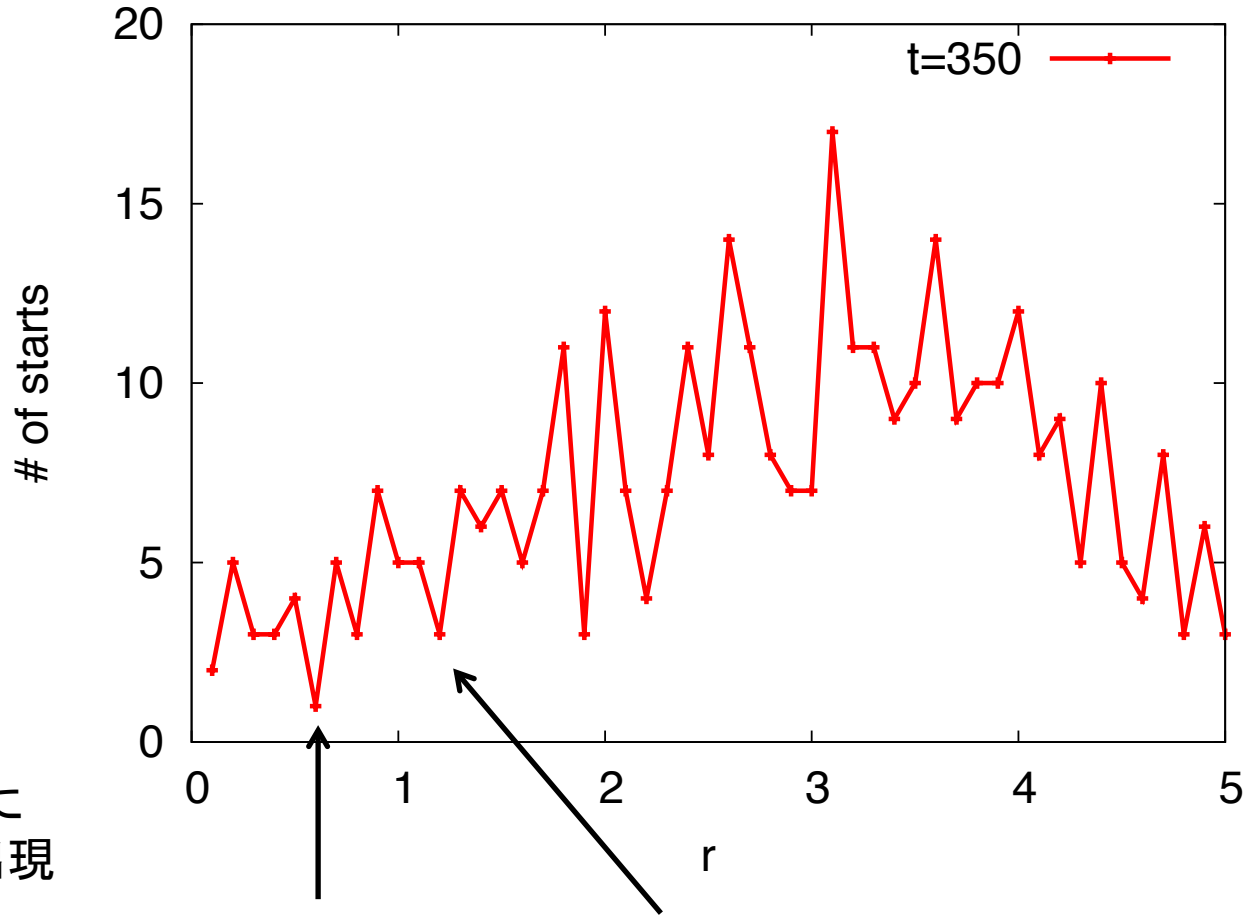


銀河の形成　#3

計算結果：

時刻350における星の頻度分布

（各恒星の半径は単純に銀河中心からの距離とした）



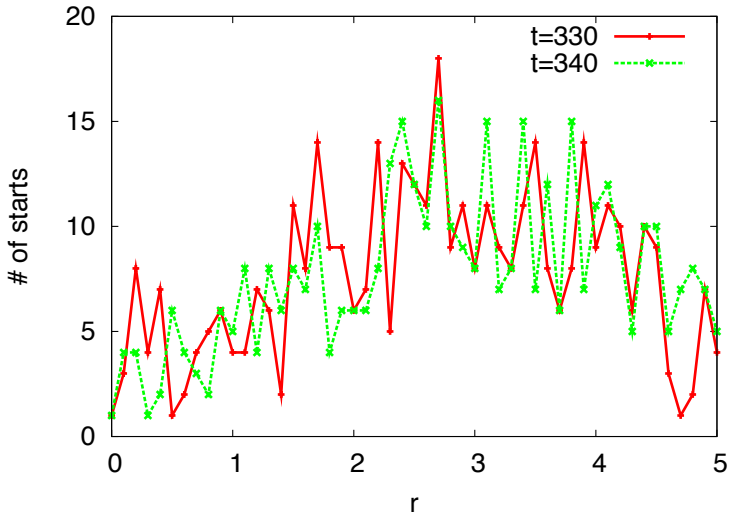
頻度分布にばらつき出現

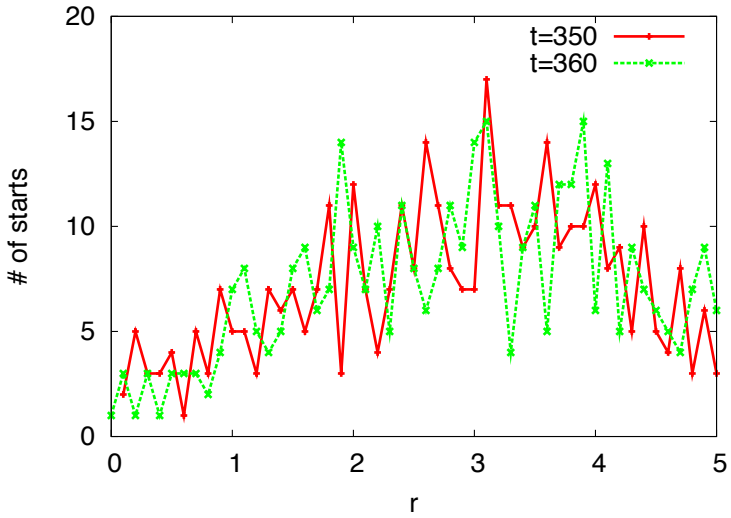
周期比1:6に近い？

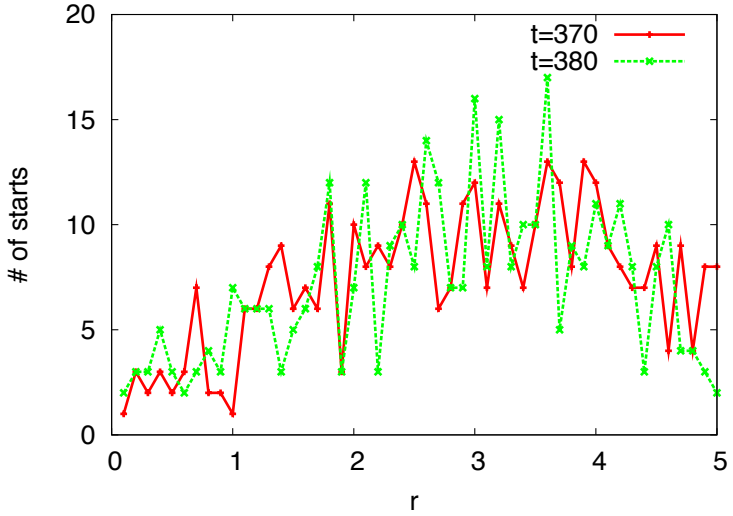
周期比1:2に近い？

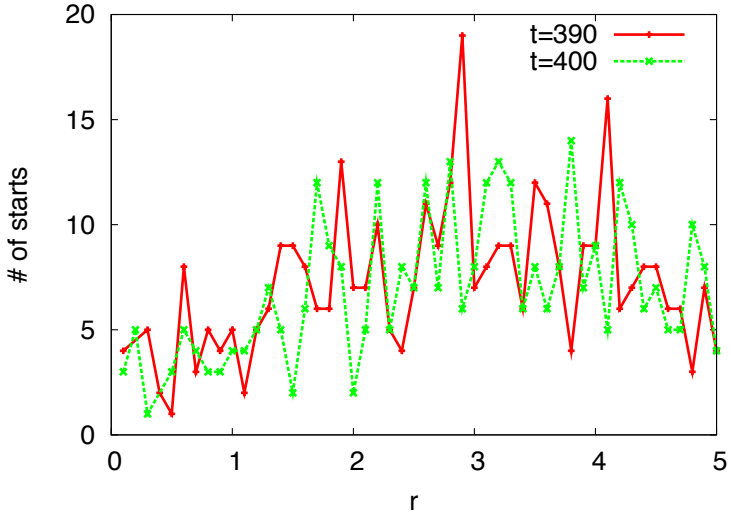
銀河の形成　#4

各時間での頻度分布



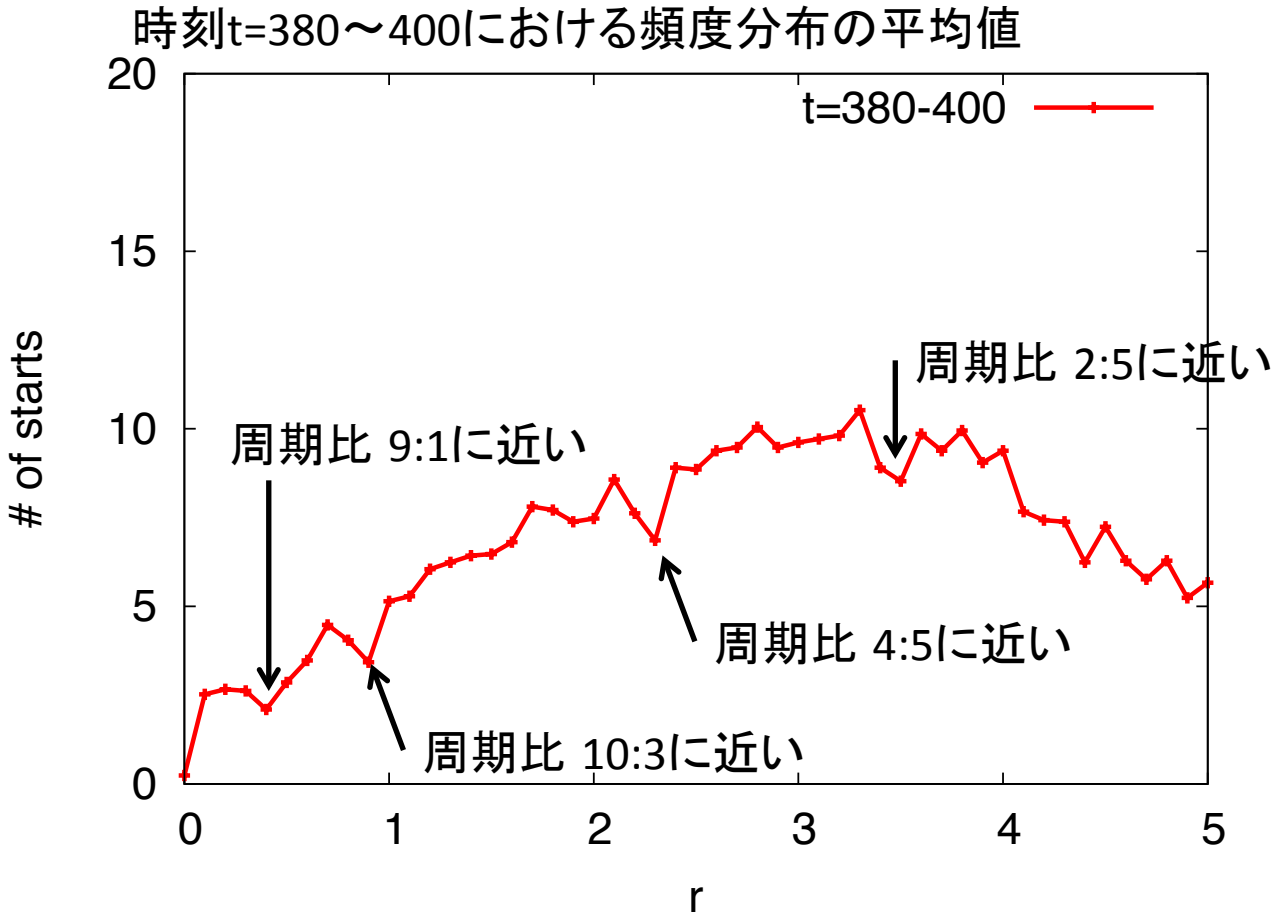






頻度分布は時間とともに変化
→恒星が楕円軌道で回転しているので、あたりまえかもしれない

銀河の形成　#5



平均化により、データがなだらかになり、凹凸が明確化
幾つかの点で出現頻度の減少を確認
これらの点は、その周期が重い星の周期と整数比関係にあり、一応出現頻度の減少は説明できる
→周期が整数比になっている他の点もあるはずなので、完全に説明できているとは言えないかもしれない？