

---

# Advanced Computational Methods for Monte Carlo Calculations

---

**NE-515-006**  
**Spring 2018**

**LA-UR-18-20247**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE**  
**Senior R&D Scientist, Monte Carlo, LANL**

 **NUCLEAR  
ENGINEERING**

 **Los Alamos**  
NATIONAL LABORATORY  
EST. 1943

# Abstract

---

## Advanced Computational Methods for Monte Carlo Calculations

Prof. Forrest Brown

This course is intended for graduate students who already have a basic understanding of Monte Carlo methods. It focuses on advanced topics that may be needed for thesis research, for developing new state-of-the-art methods, or for working with modern production Monte Carlo codes. Topics to be covered include:

- Linear Boltzmann transport equation & integral form
- Optimal random sampling from piecewise-linear PDFs
- Parallel & vector Monte Carlo algorithms
- Green's functions, the fission matrix, and linear integral operators
- Adjoint-weighted integrals & sensitivity analysis
- Precision & roundoff considerations, IEEE-floating point
- Bit operations & random number generators
- Detailed workings of delta-tracking & 3D CSG

Thorough knowledge of some programming language is required (e.g., C++, Fortran-2003, perl, python). A previous course in transport theory is recommended. Students are assumed to be familiar with the material in UNM NE-462 / NE-562 (see F. Brown, "Monte Carlo Techniques for Nuclear Systems", LA-UR-16-29043, in the Reference Collection at the [mcnp.lanl.gov](http://mcnp.lanl.gov) website)

Meet: 3 hours/week

# Lecture Topics

---

## Transport Theory & Physics

- AMC-10 Linear Boltzmann Transport Equation & Integral Form
- AMC-11 Adjoints & Green's Functions
- AMC-12 Fission Matrix Method for MC Criticality Problems
- AMC-13 Continuously Varying Materials & Tallies

## Random Numbers & Sampling

- AMC-20 Random Number Generators & RNG Testing
- AMC-21 Random Sampling – Beyond the Basics
- AMC-22 Optimal Random Sampling from Piecewise-Linear PDFs
- AMC-23 Permutations, Sets of N-from-M, & Counting-sorts
- AMC-24 Some Ideas for a New Random Number Generator

## Code Development

- AMC-30 Monte Carlo Codes – Basic Algorithm & Structure
- AMC-31 Code Development – How to Time & Test
- AMC-32 Vector & Parallel Monte Carlo
- AMC-33 Optimizing Monte Carlo Calculations



**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# References

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**

# References - Transport Theory

---

- **Bell & Glasstone, Nuclear Reactor Theory (1970) ★**

This is one of the all-time classics for transport theory & reactor physics. Very readable at the grad-student & professional level. The focus is entirely on neutron transport. Covers the neutron transport equation, 1-speed transport theory, numerical methods ( $P_n$  & diffusion), multigroup methods, discrete ordinates methods, the adjoint equation, perturbation theory, variational methods, neutron thermalization, resonance absorption, reactor dynamics, & more. Hard-copy book can be ordered through the ANS bookstore at ans.org. PDF-copy readily available.

- **Ganapol, Analytical Benchmarks for Nuclear Engineering Applications, OECD-NEA publication (2008) ★**

This is a nice book to have, even if you don't care about the analytical benchmark solutions. The first 25 pages provide a concise overview of the neutron Boltzmann transport equation. PDF-copy readily available.

- **Cacuci (Ed), Handbook of Nuclear Engineering, Chapter 5 (by Prinja & Larsen) (2010) ★**

This is the transport theory section of the recent edition of the Handbook of Nuclear Engineering. With 116 pages, it is fairly complete, but a bit pedantic. PDF-copy available.

- **Duderstadt & Martin, Transport Theory (1979) ★**

Another classic reference. Huge amount of material, including both theory & numerical methods, PDF-copy available.

- **MMR Williams, Mathematical Methods in Particle Transport Theory (1971) ★**

PDF-copy available.

- **E.E. Lewis and W.F. Miller, Jr., Computational Methods of Neutron Transport, ANS (1993)**

# References – Monte Carlo

---

- **Available in the Reference Collection at [mcnp.lanl.gov](http://mcnp.lanl.gov)**

- **F.B. Brown, "Monte Carlo Techniques for Nuclear Systems - Theory Lectures", LA-UR-16-29043, (2016) ★**  
Lecture notes on theory for the Monte Carlo class that is taught to senior undergraduate & graduate students in the Nuclear Engineering Department at the University of New Mexico. For undergraduate students, this 1-semester class is required for graduation. There are 600 slides covering all of the basics for Monte Carlo particle transport & some advanced material. Portions of these notes are also used in the MCNP Criticality Class given at LANL. (2016)
- **X-5 Monte Carlo Team, "MCNP - A General N-Particle Transport Code, Version 5" Volume I: Overview and Theory, LA-UR-03-1987 (2003, updated 2005) ★**
- **L.L. Carter and E.D. Cashwell, Particle Transport Simulation with the Monte Carlo Method, ERDA Critical Review Series, TID-26607, National Technical Information Service, Springfield MA (1975). ★**
- **A. Sood, "The Monte Carlo Method and MCNP - A Brief Review of Our 40 Year History", Int. Topical Meeting on Industrial Radiation and Radioisotope Measurement - Applications Conference, Chicago IL, July, LA-UR-17-26533 (2017). ★**
- **E.D. Cashwell and C.J. Everett, A Practical Manual on the Monte Carlo Method for Random Walk Problems, Pergamon Press, London LA-2120, (1959). ★**
- **H. Kahn, "Applications of Monte Carlo," Rand Corporation, Santa Monica, CA (1954), AECU-3259 ★**
- **R.C. Gast and N.R. Candelore, "Monte Carlo Eigenfunction Strategies and Uncertainties," in Proc. NEACRP Meeting of a Monte Carlo Study Group, ANL-75-2, Argonne National Laboratory, Argonne, IL (1974).**
- **E.M. Gelbard and R.E. Prael, "Monte Carlo Work at Argonne National Laboratory", in Proc. NEACRP Meeting of a Monte Carlo Study Group, ANL-75-2, Argonne National Laboratory, Argonne, IL (1974).**

- **Other**

- **Lux & L. Koblinger, Monte Carlo Particle Transport Methods: Neutron and Photon Calculations, CRC Press, Boston (1991). ★**

# References – Random Numbers & Random Sampling

---

- D. E. Knuth, The Art of Computer Programming, Vol. 2: Semi-numerical Algorithms, 3<sup>rd</sup> Edition, Addison-Wesley, Reading, MA (1998).
- L. Devroye, Non-Uniform Random Variate Generation, Springer-Verlag, NY (1986). ★
- J. von Neumann, "Various Techniques Used in Conjunction with Random Digits," *J. Res. Nat. Bur. Stand. Appl. Math Series* 3, 36-38 (1951). ★
- C. J. Everett and E. D. Cashwell, "A Third Monte Carlo Sampler," LA9721-MS, Los Alamos National Laboratory, Los Alamos, NM (1983). ★
- H. Kahn, "Applications of Monte Carlo," AECU-3259, Rand Corporation, Santa Monica, CA (1954). ★
- F.B. Brown, "Random Number Generation with Arbitrary Strides", *Trans. Am. Nucl. Soc.* (Dec 1994)
- F.B. Brown & Y. Nagaya, "The MCNP5 Random Number Generator", *Trans. Am. Nucl. Soc.* [also, LA-UR-02-3782] (November, 2002) ★
- Y. Nagaya & F.B. Brown, "Testing MCNP Random Number Generators", LANL report on testing MCNP5 RN generators, work performed in 2002 for original MCNP5 version, LA-UR-11-04858 (2011) ★

# NE-515-006

## Course Information

**NE-515-006**  
**Spring 2018**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE**  
**Senior R&D Scientist, Monte Carlo, LANL**



# Advanced Computational Methods for Monte Carlo Calculations

---

This course is intended for graduate students and professionals who already have a basic understanding of Monte Carlo methods. It focuses on advanced topics that may be needed for thesis research, for developing new state-of-the-art methods, or for working with modern production Monte Carlo codes.

Thorough knowledge of some programming language is required (e.g., C++, C, Fortran-2003, perl, python, Matlab). A previous course in transport theory is recommended. Students are assumed to be familiar with the material in UNM NE-462 / NE-562 (see F. Brown, "Monte Carlo Techniques for Nuclear Systems", LA-UR-16-29043, in the Reference Collection at the [mcnp.lanl.gov](http://mcnp.lanl.gov) website). Meet 3 hours/week.

## Lecture Topics:

### Transport Theory & Physics

- AMC-10 Linear Boltzmann Transport Equation & Integral Form
- AMC-11 Adjoints & Green's Functions
- AMC-12 Fission Matrix Method for MC Criticality Problems
- AMC-13 Continuously Varying Materials & Tallies

### Random Numbers & Sampling

- AMC-20 Random Number Generators & RNG Testing
- AMC-21 Random Sampling – Beyond the Basics
- AMC-22 Optimal Random Sampling from Piecewise-Linear PDFs
- AMC-23 Permutations, Sets of N-from-M, & Counting-sorts

### Code Development

- AMC-30 Monte Carlo Codes – Basic Algorithm & Structure
- AMC-31 Code Development – How to Time & Test
- AMC-32 Vector & Parallel Monte Carlo
- AMC-33 Optimizing Monte Carlo Calculations

# NE-515-006 Information

---

- **Focus – advanced Monte Carlo methods**

- Transport theory
- Random sampling
- Coding issues

Target audience is (1) graduate students who may need to write their own MC codes as part of their research & (2) professionals who need to know the underlying theory & sampling methods that arise in mature, production-level MC codes.

This is not a beginning course in MC methods. Students are assumed to be familiar with the basics of MC methods, as in UNM NE-462 / NE-562 (see F. Brown, "Monte Carlo Techniques for Nuclear Systems", LA-UR-16-29043, in the Reference Collection at the [mcnp.lanl.gov](http://mcnp.lanl.gov) website)

Production MC codes such as MCNP are not used or required. There is no discussion of using MCNP or preparing MCNP input for application problems.

Some computer programming is required. Any language is OK (Preferred: C++, C, Fortran-2003, python, perl; Acceptable: Matlab)

- **Office hours, discussion, help**

- Wednesdays – about 1 hour before/after classes
- Email – anytime, 7:00-4:00 - [fbrown@lanl.gov](mailto:fbrown@lanl.gov), other times – [fbrown@q.com](mailto:fbrown@q.com)
- Other office hours by request

- **Grading**

- There are a few homework assignments. These will be discussed in class & not graded
- Attendance at most classes is expected
- A project is required & graded. One of the following:
  - A MC code or calculations that directly support your research. Send a 1-paragraph description.
  - Write a 3D, multigroup, mesh-based MC code. Specific features & tests will be discussed.

# UNM NE-515-006, Spring 2018

---

- **Schedule**

- Lecture topics will vary among transport, sampling, & codes. Depending on class interests, additional topics are possible.
- Rough schedule is:

1/17	AMC-10	Linear Boltzmann Transport Equation & Integral Form
1/24	AMC-30	Monte Carlo Codes – Basic Algorithm & Structure
	AMC-31	Code Development – How to Time & Test
1/31	AMC-20	Random Number Generators & RNG Testing
2/7	AMC-21	Random Sampling – Beyond the Basics
2/14	AMC-22	Optimal Random Sampling from Piecewise-Linear PDFs
2/21	AMC-23	Permutations, Sets of N-from-M, & Counting-sorts
2/28	AMC-11	Adjoint & Green's Functions
3/7	AMC-12	Fission Matrix Method for MC Criticality Problems
3/14		break
3/21	AMC-33	Optimizing Monte Carlo Calculations
3/28	AMC-32	Vector & Parallel Monte Carlo
4/4	AMC-13	Continuously Varying Materials & Tallies
4/11		Project presentations &/or discussion
4/18		Project presentations &/or discussion
4/25		Project presentations &/or discussion
5/2		Project presentations &/or discussion

**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **The Linear Boltzmann Transport Equation**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Outline

---

- **Introduction**
- **Assumptions**
- **Linear Boltzmann Transport Equation**
- **Integral Form & Basis for Monte Carlo Simulation**
- **Monte Carlo Eigenvalue Problems**

---

# Introduction

# Introduction

---

- **Monte Carlo methods (MC) can be used to simulate the transport of radiation through matter**
  - These lectures will focus on neutral particles (e.g., neutrons & photons)
  - It will also be assumed that we are solving linear problems, where the material properties and geometry are fixed during the MC simulation
- **The fundamental equation being solved is the linear Boltzmann transport equation (LBTE)**
  - We will focus on interpreting & using the LBTE, not deriving it
- **Reading**
  1. Bell & Glasstone, Nuclear Reactor Theory, pp 1-20, 21-27, 35-37
  2. Ganapol, Analytical Benchmarks for Nuclear Engineering Applications, pp 1-14
  3. Cacuci, Handbook of Nuclear Engineering, Chapter 5 (Prinja & Larsen), pp 430-464

# Introduction

---

- **The LBTE provides a continuum description of the behavior of radiation particles in matter**
  - For a given radiation source, the solution of the LBTE gives the angular flux,  $\psi(r,E,\Omega,t)$ , a continuous function (or field)
  - $\psi(r,E,\Omega,t)$  represents the average behavior of a very, very large number of particles (in nature, typically  $10^4 - 10^{18}$  particles/cm<sup>3</sup>)
  - Physical results are obtained by integrating  $\psi(r,E,\Omega,t)$  with some response function:

$$\text{fission rate} = \iiint_{V,E,\Omega,t} dr dE d\Omega dt \quad \Sigma_F(r,E) \cdot \psi(r,E,\Omega,t)$$

- **LBTE describes continuum, but MC simulates discrete particles**
- **MC simulates the behavior of individual particles**
  - To obtain a solution to the LBTE, must simulate very many particles
  - Average behavior of the particles gives  $\psi(r,E,\Omega,t)$  (with uncertainty)
  - In the limit of many particles, MC average results approach  $\psi(r,E,\Omega,t)$

# Introduction

---

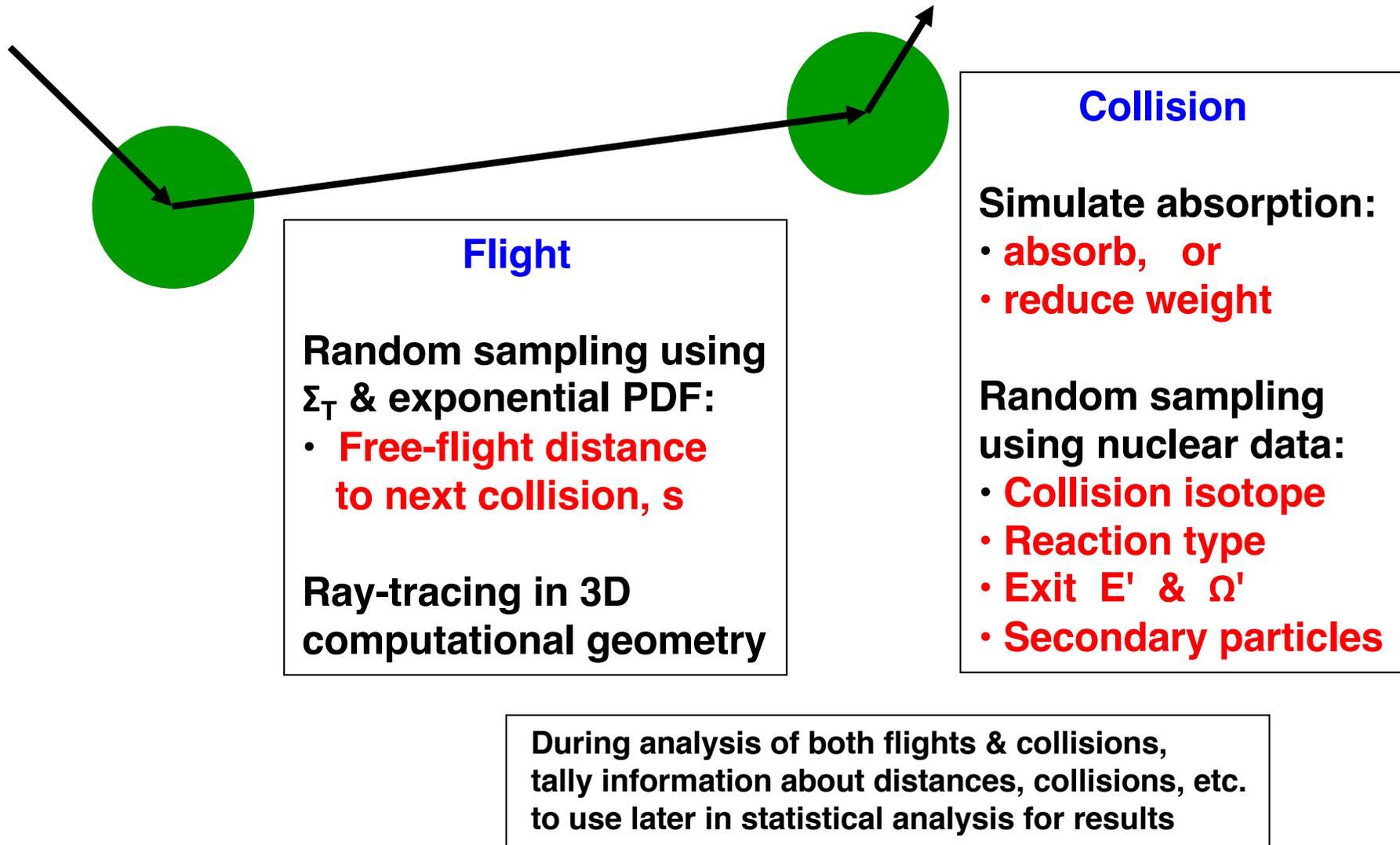
- **The LBTE is an integro-differential equation**
- **MC methods compute integrals (or averages)**
- **General approach in what follows:**
  - **Examine the LBTE, including what every term represents**
  - **Convert the LBTE integro-differential equation into an integral form**
  - **Examine the integral LBTE to see the fundamental basis for the MC solution**
  - **Consider time-independent steady state cases – k-eigenvalue &  $\alpha$ -eigenvalue forms of the LBTE**
  - **In some later lectures...**
    - **Start over, defining & using a Green's function approach**
    - **Introduce the adjoint transport equation**

---

# Monte Carlo Simulation & Assumptions

# Monte Carlo Simulation of Radiation Transport

- **Goal:** Simulate nature, particles moving through physical objects



# Assumptions for LBTE & MC Simulation

---

## Assume:

- Neutrons & photons are particles, not waves
- Particles move in a straight line between collisions (neutrons, photons)
- Collisions occur instantaneously, at a point in space
- Ignore neutron-neutron collisions
  
- Particle speeds are small enough to neglect relativistic effects
- Particle speeds are high enough to neglect quantum effects
  
- Particle collisions do not change the properties of a material (ie, no feedback, no material heating, no depletion)
- Material properties are fixed for the duration of the simulation (geometry, densities, temperatures, material compositions, .....

## Why?

- Want to solve the linear Boltzmann transport equation
- Want to apply the superposition principle
- Want the Central Limit Theorem to apply for computing statistics
  - Statisticians love the term “IID” - Independent, Identically Distributed

(Any or all of the above assumptions can be relaxed, with careful analysis & extra computing cost.)

# Linear Boltzmann Transport Equation

# Linear Boltzmann Transport Equation

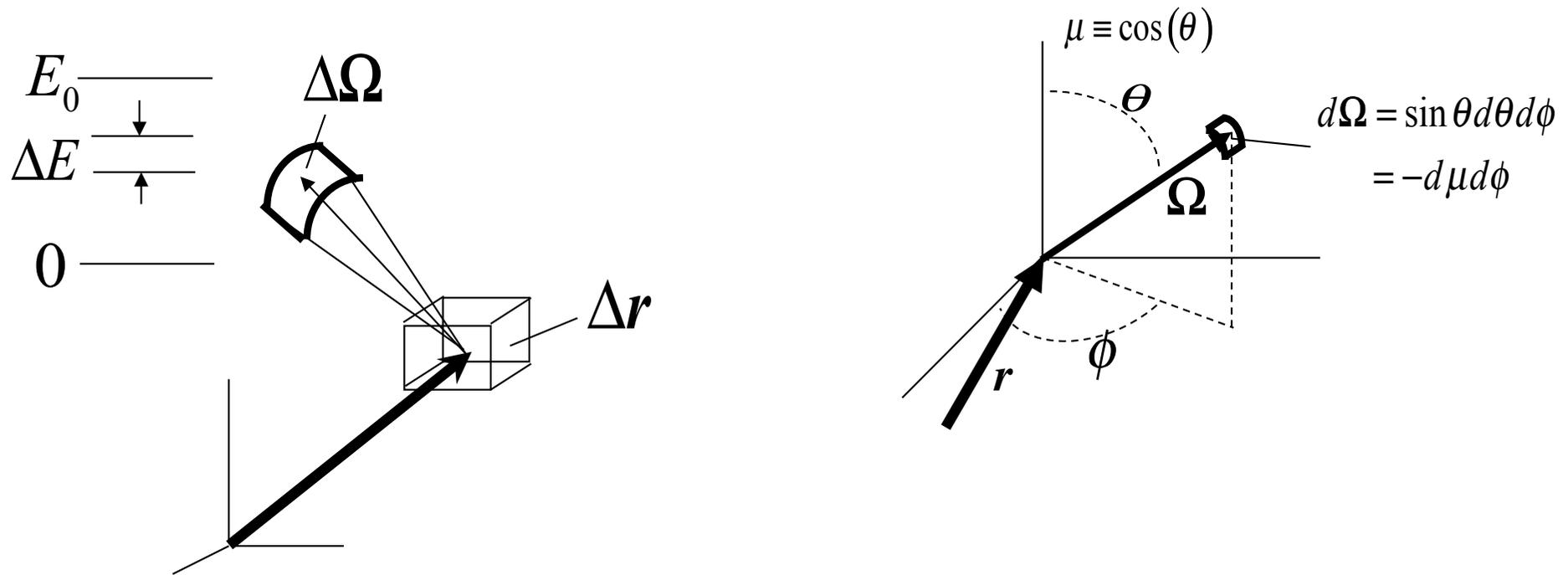
- Time-dependent linear Boltzmann transport equation for neutrons, with prompt fission source & external source

$$\begin{aligned}
 \frac{1}{v} \frac{\partial \psi(\vec{r}, E, \vec{\Omega}, t)}{\partial t} = & \quad \text{External source} \quad Q(\vec{r}, E, \vec{\Omega}, t) + \quad \text{Scattering} \quad \iint \psi(\vec{r}, E', \vec{\Omega}', t) \Sigma_s(\vec{r}, E' \rightarrow E, \vec{\Omega} \cdot \vec{\Omega}') d\vec{\Omega}' dE' \\
 & + \quad \text{Multiplication} \quad \frac{\chi(\vec{r}, E)}{4\pi} \iint v \Sigma_f(\vec{r}, E') \psi(\vec{r}, E', \vec{\Omega}', t) d\vec{\Omega}' dE' \\
 & - \quad \text{Leakage} \quad \text{Collisions} \quad \left[ \vec{\Omega} \cdot \nabla + \Sigma_T(\vec{r}, E) \right] \cdot \psi(\vec{r}, E, \vec{\Omega}, t)
 \end{aligned}$$

$$\frac{1}{v} \frac{\partial \psi(\vec{r}, E, \vec{\Omega}, t)}{\partial t} = \underbrace{Q}_{\text{Gains}} + \underbrace{[S+M]}_{\text{Gains}} \cdot \psi - \underbrace{[L+T]}_{\text{Losses}} \cdot \psi$$

- This equation can be solved directly by Monte Carlo, assuming:
  - Each neutron history is an IID trial (independent, identically distributed)
  - All neutrons must see same probability densities in all of phase space
  - Usual method: geometry & materials fixed over solution interval  $\Delta t$

## The LBTE is a Balance Equation



- Contributions to the Total Neutron Balance during  $\Delta t$**

$$\begin{aligned}
 \text{Number in } \Delta r \Delta \Omega \Delta E \text{ at } t + \Delta t &= \text{Number in } \Delta r \Delta \Omega \Delta E \text{ at } t \\
 &+ \text{Number **gained** during } \Delta t \\
 &- \text{Number **lost** during } \Delta t
 \end{aligned}$$

Note: for this discussion, we will assume all neutrons are prompt

$$\frac{1}{v} \cdot \frac{\partial}{\partial t} \psi(\vec{r}, E, \vec{\Omega}, t) = Q + [S + M] \cdot \psi - [L + T] \cdot \psi$$


---

$$\boxed{\text{Number in } \Delta r \Delta \Omega \Delta E \text{ at } t + \Delta t} = \Delta \Omega \Delta E \int_V dr n(r, \Omega, E, t + \Delta t)$$

$$\boxed{\text{Number in } \Delta r \Delta \Omega \Delta E \text{ at } t} = \Delta \Omega \Delta E \int_V dr n(r, \Omega, E, t)$$

$$\psi(\vec{r}, E, \hat{\Omega}, t) = v n(\vec{r}, E, \hat{\Omega}, t)$$

$$n(\vec{r}, E, \hat{\Omega}, t) = \frac{1}{v} \psi(\vec{r}, E, \hat{\Omega}, t)$$

$$\frac{\partial n(\vec{r}, E, \hat{\Omega}, t)}{\partial t} = \frac{1}{v} \frac{\partial \psi(\vec{r}, E, \hat{\Omega}, t)}{\partial t}$$

$$\frac{1}{v} \cdot \frac{\partial}{\partial t} \psi(\vec{r}, E, \vec{\Omega}, t) = \mathbf{Q} + [\mathbf{S} + \mathbf{M}] \cdot \psi - [\mathbf{L} + \mathbf{T}] \cdot \psi$$


---

- **Source term, Q**

- **Accounts for particles added from some external source, not from scatter or fission within system**
- **May be an internal source – point, line, volume source**
  - Particles added to  $\Delta r \Delta \Omega \Delta E$  during  $\Delta t$

$$\Delta \Omega \Delta E \Delta t \int dr Q( r, \Omega, E, t )$$

- **May be an incoming boundary source**
  - Particles added to  $\Delta r \Delta \Omega \Delta E$  on boundary during  $\Delta t$

$$\Delta \Omega \Delta E \Delta t \int dr Q( r, \Omega, E, t ) \delta(r-r_s)$$

$$\frac{1}{V} \cdot \frac{\partial}{\partial t} \Psi(\vec{r}, E, \vec{\Omega}, t) = Q + [\mathbf{S} + \mathbf{M}] \cdot \Psi - [\mathbf{L} + \mathbf{T}] \cdot \Psi$$

**Number gained in  $\Delta r \Delta \Omega \Delta E$   
from scattering during  $\Delta t$**

$$= \Delta \Omega \Delta E \Delta t \int_V d\mathbf{r} \int_0^\infty dE' \int_{4\pi} d\Omega' \underbrace{f_s(\Omega' \cdot \Omega, E' \rightarrow E)}_{\text{Probability of scatter from } \Omega', E' \text{ to } \Omega, E} \underbrace{\Sigma_s(\mathbf{r}, E') \Psi(\mathbf{r}, \Omega', E', t)}_{\text{number scattering at } \Omega', E'}$$

**Probability of scatter from  $\Omega', E'$  to  $\Omega, E$**

- **Joint pdf for  $E, \Omega$  exiting collision**
- **For some types of scatter, may be factored as  $f_\mu(\mu) f_E(E|\mu)$**
- **Angular dependence of scattering from  $E', \Omega'$  to  $E, \Omega$  depends on the cosine of the scattering angle  $\Omega' \cdot \Omega$ , not the individual directions**

**number scattering  
at  $\Omega', E'$**

$$\frac{1}{v} \cdot \frac{\partial}{\partial t} \psi(\vec{r}, E, \vec{\Omega}, t) = Q + [S + M] \cdot \psi - [L + T] \cdot \psi$$

Number gained in  
 $\Delta r \Delta \Omega \Delta E$  from  
fission during  $\Delta t$

$$= \Delta \Omega \Delta E \Delta t \int_V d\mathbf{r} \frac{\chi(E)}{4\pi} \int_0^\infty dE' \int_{4\pi} d\Omega' v(E') \Sigma_f(\mathbf{r}, E') \psi(\mathbf{r}, \Omega', E', t)$$

Probability of neutrons being  
produced in  $\Delta \Omega \Delta E$  during  $\Delta t$

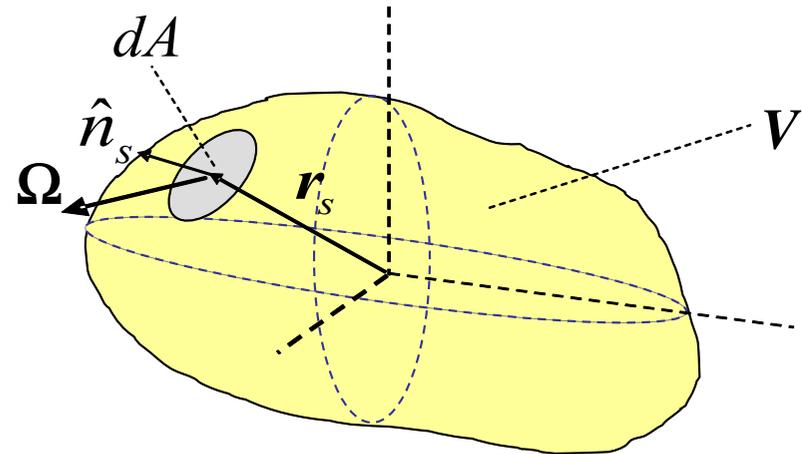
$\chi$  = pdf for energy  $E$  of fission  
neutrons produced

$1/4\pi$  = isotropic emission in  $\Omega$

fissions due to  
neutrons at  $E', \Omega'$

average number of  
neutrons per fission

$$\frac{1}{v} \cdot \frac{\partial}{\partial t} \psi(\vec{r}, E, \vec{\Omega}, t) = Q + [S + M] \cdot \psi - [L + T] \cdot \psi$$



$$\mathbf{J}(r, \Omega, E, t) = \Omega \psi(r, \Omega, E, t)$$

Number lost through  
surface  $dA$  of  $V$

$$= \Delta \Omega \Delta E \left[ \Delta t \left\{ \hat{n}_s \cdot \mathbf{J}(r, \Omega, E, t) dA \right\} \right]$$

Number lost through entire  
surface of  $\Delta r \Delta \Omega \Delta E$  during  $\Delta t$

$$\begin{aligned} &= \Delta \Omega \Delta E \Delta t \int_A dA \hat{n}_s \cdot \mathbf{J}(r_s, \Omega, E, t) = \Delta \Omega \Delta E \Delta t \int_V dr \nabla \cdot \mathbf{J}(r, \Omega, E, t) \\ &= \Delta \Omega \Delta E \Delta t \int_V dr \Omega \cdot \nabla \psi(r, \Omega, E, t) \end{aligned}$$

**note: Application of divergence theorem**

$$\frac{1}{v} \cdot \frac{\partial}{\partial t} \psi(\vec{r}, E, \vec{\Omega}, t) = Q + [S + M] \cdot \psi - [L + T] \cdot \psi$$

Number lost through absorption in  $\Delta r \Delta \Omega \Delta E$  during  $\Delta t$   $= \Delta \Omega \Delta E \Delta t \int_V d\mathbf{r} \Sigma_a(\mathbf{r}, E) \psi(\mathbf{r}, \Omega, E, t)$

Number scattering out of  $\Delta \Omega \Delta E$  during  $\Delta t$

$$= \int_V d\mathbf{r} \left[ \int_0^\infty \int_{4\pi} f_s(\Omega' \bullet \Omega, E \rightarrow E') dE' d\Omega' \right] \Sigma_s(\mathbf{r}, E) \psi(\mathbf{r}, \Omega, E, t) \Delta \Omega \Delta E \Delta t$$

1

Number in  $\Delta r \Delta \Omega \Delta E$  lost to absorption & scatter-out during  $\Delta t$

$$= \Delta \Omega \Delta E \Delta t \int_V d\mathbf{r} \left[ \Sigma_A(\mathbf{r}, E) + \Sigma_s(\mathbf{r}, E) \right] \psi(\mathbf{r}, \Omega, E, t)$$

$$= \Delta \Omega \Delta E \Delta t \int_V d\mathbf{r} \Sigma_T(\mathbf{r}, E) \psi(\mathbf{r}, \Omega, E, t)$$

$$\frac{1}{v} \cdot \frac{\partial}{\partial t} \psi(\vec{r}, E, \vec{\Omega}, t) = Q + [S + M] \cdot \psi - [L + T] \cdot \psi$$

**Neutron balance equation in V:**

$$\begin{aligned} \Delta t &\rightarrow 0 \\ \Delta \Omega &\rightarrow 0 \\ \Delta E &\rightarrow 0 \end{aligned}$$

$$\int_V d\mathbf{r} \left\{ \begin{aligned} &\left[ \frac{1}{v} \frac{\partial}{\partial t} + \boldsymbol{\Omega} \cdot \nabla + \Sigma(r, E) \right] \psi(r, \boldsymbol{\Omega}, E, t) - Q(r, \boldsymbol{\Omega}, E, t) \\ &- \int_0^\infty dE' \int_{4\pi} d\boldsymbol{\Omega}' \Sigma_s(r, \boldsymbol{\Omega}' \cdot \boldsymbol{\Omega}, E' \rightarrow E) \psi(r, \boldsymbol{\Omega}', E', t) - \\ &- \frac{\chi(E)}{4\pi} \int_0^\infty dE' \int_{4\pi} d\boldsymbol{\Omega}' v(E') \Sigma_f(r, E') \psi(r, \boldsymbol{\Omega}', E', t) \end{aligned} \right\} = 0$$

$$\frac{1}{v} \cdot \frac{\partial}{\partial t} \psi(\vec{r}, E, \vec{\Omega}, t) = Q + [S + M] \cdot \psi - [L + T] \cdot \psi$$


---

**Implies the Time Dependent Neutron Transport Equation:**

$$\left[ \frac{1}{v} \frac{\partial}{\partial t} + \mathbf{\Omega} \cdot \nabla + \Sigma(r, E) \right] \psi(r, \mathbf{\Omega}, E, t) =$$

$$= \int_0^{\infty} dE' \int_{4\pi} d\mathbf{\Omega}' \Sigma_s(r, \mathbf{\Omega}' \cdot \mathbf{\Omega}, E' \rightarrow E) \psi(r, \mathbf{\Omega}', E', t) +$$

$$+ \frac{\chi(E)}{4\pi} \int_0^{\infty} dE' \int_{4\pi} d\mathbf{\Omega}' v(E') \Sigma_f(r, E') \psi(r, \mathbf{\Omega}', E', t) + Q(r, \mathbf{\Omega}, E, t)$$

# Linear Boltzmann Transport Equation

- **Time-dependent linear Boltzmann transport equation for neutrons, with prompt fission source & external source**

$$\begin{aligned}
 \frac{1}{v} \frac{\partial \psi(\vec{r}, E, \vec{\Omega}, t)}{\partial t} = & \overset{\text{External source}}{Q(\vec{r}, E, \vec{\Omega}, t)} + \overset{\text{Scattering}}{\iint \psi(\vec{r}, E', \vec{\Omega}', t) \Sigma_s(\vec{r}, E' \rightarrow E, \vec{\Omega} \cdot \vec{\Omega}') d\vec{\Omega}' dE'} \\
 & + \overset{\text{Multiplication}}{\frac{\chi(\vec{r}, E)}{4\pi} \iint v \Sigma_f(\vec{r}, E') \psi(\vec{r}, E', \vec{\Omega}', t) d\vec{\Omega}' dE'} \\
 & - \overset{\text{Leakage}}{[\vec{\Omega} \cdot \nabla]} - \overset{\text{Collisions}}{\Sigma_T(\vec{r}, E)} \cdot \psi(\vec{r}, E, \vec{\Omega}, t)
 \end{aligned}$$

$$\frac{1}{v} \frac{\partial \psi(\vec{r}, E, \vec{\Omega}, t)}{\partial t} = \underset{\text{Gains}}{Q + [S + M] \cdot \psi} - \underset{\text{Losses}}{[L + T] \cdot \psi}$$

- **This equation can be solved directly by Monte Carlo, assuming:**
  - Each neutron history is an IID trial (independent, identically distributed)
  - All neutrons must see same probability densities in all of phase space
  - **Usual method: geometry & materials fixed over solution interval  $\Delta t$**

# **Integral Transport Equation & Basis for MC Simulation**

# Monte Carlo & Transport Equation

- **Derive integral equation, in kernel form**

- Start with integro-differential equation

- Use integrating factor

$$\exp\left[-\int_0^R \Sigma_T(\vec{r} - R\hat{\Omega}, \mathbf{E}) dR'\right], \quad \text{where } R\hat{\Omega} = \vec{r} - \vec{r}'$$

- Define  $\vec{\mathbf{E}} = \mathbf{E} \cdot \hat{\Omega}$

**Collision density:**  $\Psi(\vec{r}, \vec{\mathbf{E}}) = \Sigma_T(\vec{r}, \mathbf{E}) \cdot \psi(\vec{r}, \vec{\mathbf{E}})$

**Transport kernel:**  $\mathbf{T}(\vec{r}' \rightarrow \vec{r}, \vec{\mathbf{E}}) = \Sigma_T(\vec{r}, \mathbf{E}) \cdot \exp\left[-\int_0^{|\vec{r}-\vec{r}'|} \Sigma_T(\vec{r}' + s\hat{\Omega}, \mathbf{E}) ds\right] \cdot \frac{\delta(\hat{\Omega} \cdot \frac{\vec{r}-\vec{r}'}{|\vec{r}-\vec{r}'|} - 1)}{|\vec{r}-\vec{r}'|^2}$

**Collision kernel:**  $\mathbf{C}(\vec{\mathbf{E}}' \rightarrow \vec{\mathbf{E}}, \vec{r}) = \frac{\Sigma_S(\vec{r}, \vec{\mathbf{E}}' \rightarrow \vec{\mathbf{E}})}{\Sigma_T(\vec{r}, \mathbf{E}')} + \frac{\chi(\vec{r}, \mathbf{E}) \nu \Sigma_F(\vec{r}, \mathbf{E}')}{4\pi \cdot \Sigma_T(\vec{r}, \mathbf{E}')}$

- Then

$$\Psi(\vec{r}, \vec{\mathbf{E}}) = \int \left[ \int \Psi(\vec{r}', \vec{\mathbf{E}}') \cdot \mathbf{C}(\vec{\mathbf{E}}' \rightarrow \vec{\mathbf{E}}, \vec{r}') d\vec{\mathbf{E}}' + \mathbf{Q}(\vec{r}', \vec{\mathbf{E}}') \right] \cdot \mathbf{T}(\vec{r}' \rightarrow \vec{r}, \vec{\mathbf{E}}) d\vec{r}'$$

# Monte Carlo & Transport Equation

---

## Basis for the Monte Carlo Solution Method

$$\Psi(\vec{r}, \vec{E}) = \int \left[ \int \Psi(\vec{r}', \vec{E}') \cdot C(\vec{E}' \rightarrow \vec{E}, \vec{r}') d\vec{E}' + Q(\vec{r}', \vec{E}') \right] \cdot T(\vec{r}' \rightarrow \vec{r}, \vec{E}) d\vec{r}'$$

Let  $p = (\vec{r}, \vec{E})$       and       $R(p' \rightarrow p) = C(\vec{E}' \rightarrow \vec{E}, \vec{r}') \cdot T(\vec{r}' \rightarrow \vec{r}, \vec{E})$

Expand  $\Psi$  into components,  $k$ , having 0,1,2,... collisions

$$\Psi(p) = \sum_{k=0}^{\infty} \Psi_k(p), \quad \text{with } \Psi_0(p) = \int Q(\vec{r}', \vec{E}) \cdot T(\vec{r}' \rightarrow \vec{r}, \vec{E}) d\vec{r}'$$

By definition,

$$\Psi_k(p) = \int \Psi_{k-1}(p') \cdot R(p' \rightarrow p) dp'$$

**Markovian:** collision  $k$  depends only on the results of collision  $k-1$ ,  
and not on any prior collisions  $k-2, k-3, \dots$

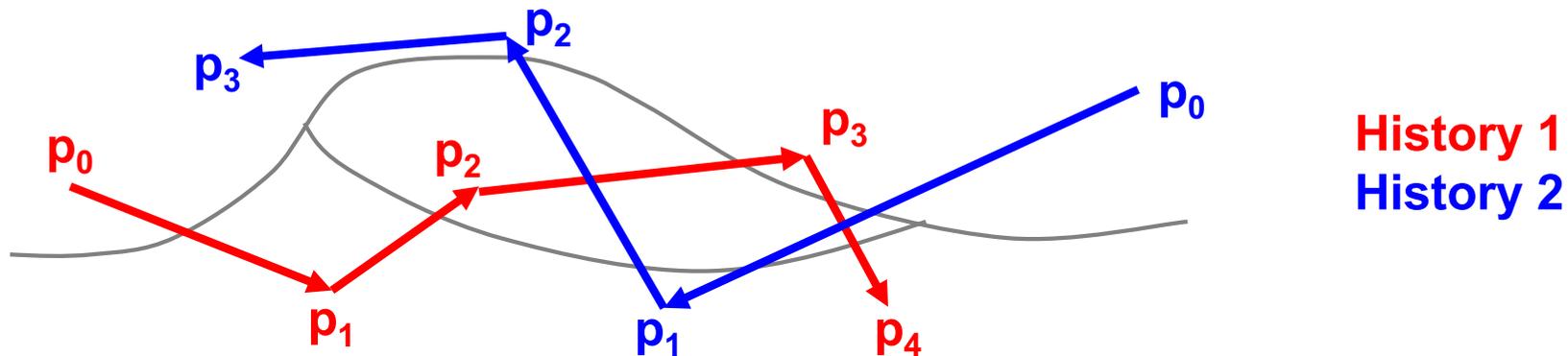
# Monte Carlo & Transport Equation

## Histories

- After repeated substitution for  $\Psi_k$

$$\begin{aligned}\Psi_k(\mathbf{p}) &= \int \Psi_{k-1}(\mathbf{p}') \cdot R(\mathbf{p}' \rightarrow \mathbf{p}) d\mathbf{p}' \\ &= \int \dots \int \Psi_0(\mathbf{p}_0) \cdot R(\mathbf{p}_0 \rightarrow \mathbf{p}_1) \cdot R(\mathbf{p}_1 \rightarrow \mathbf{p}_2) \dots R(\mathbf{p}_{k-1} \rightarrow \mathbf{p}) d\mathbf{p}_0 \dots d\mathbf{p}_{k-1}\end{aligned}$$

- A "history" is a sequence of states  $(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \dots)$



- For estimates in a given region, tally the occurrences for each collision of each "history" within a region

# Monte Carlo & Transport Equation

## Monte Carlo approach:

$$\Psi_k(\mathbf{p}) = \int \dots \int \Psi_0(\mathbf{p}_0) \cdot R(\mathbf{p}_0 \rightarrow \mathbf{p}_1) \cdot R(\mathbf{p}_1 \rightarrow \mathbf{p}_2) \dots R(\mathbf{p}_{k-1} \rightarrow \mathbf{p}) d\mathbf{p}_0 \dots d\mathbf{p}_{k-1}$$

sample  
 $\mathbf{p}_0$  →  $\mathbf{p}_1$  →  $\mathbf{p}_2$  →  $\mathbf{p}$

- For 1 trial, generate a sequence of states  $(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \dots)$  by:

- Randomly sample from PDF for source:  $\Psi_0(\mathbf{p}_0)$
- Randomly sample from PDF for  $k^{\text{th}}$  transition:  $R(\mathbf{p}_{k-1} \rightarrow \mathbf{p}_k)$
- Repeat sampling transitions until termination

- Repeat for  $M$  trials (histories)

- Generate estimates of results by averaging over states for  $M$  histories:

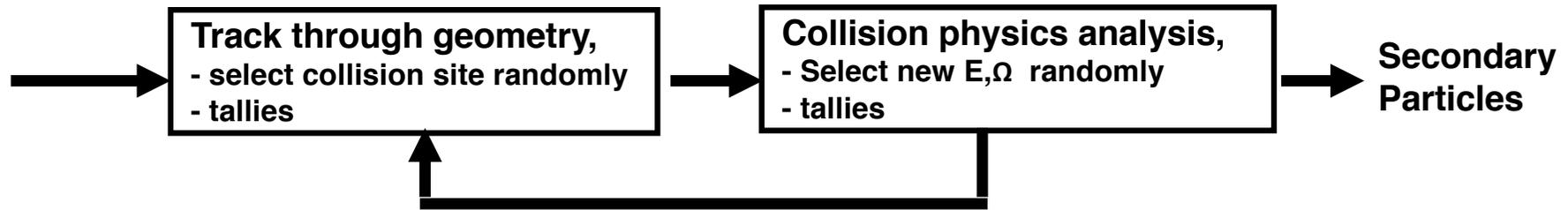
$$A = \int A(\mathbf{p}) \cdot \Psi(\mathbf{p}) d\mathbf{p} \approx \frac{1}{M} \cdot \sum_{m=1}^M \left( \sum_{k=1}^{\infty} A(\mathbf{p}_{k,m}) \right)$$

↑ Histories  
In problem

↑ Events  
In history

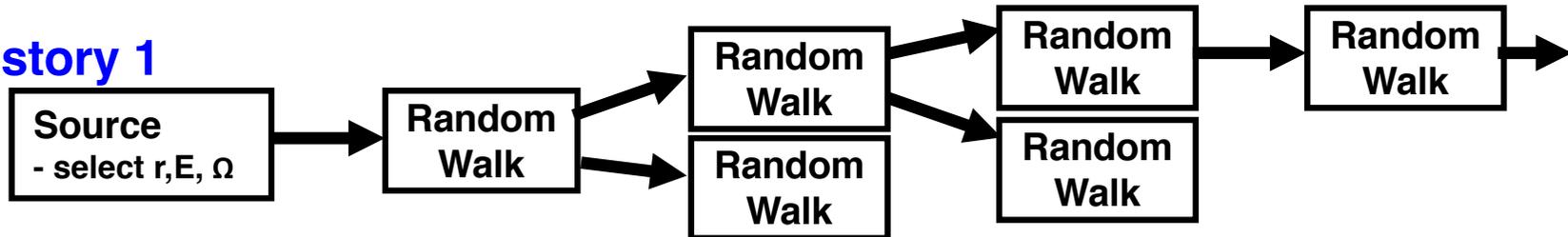
# Fixed-source Monte Carlo Calculation

## Random Walk for a particle

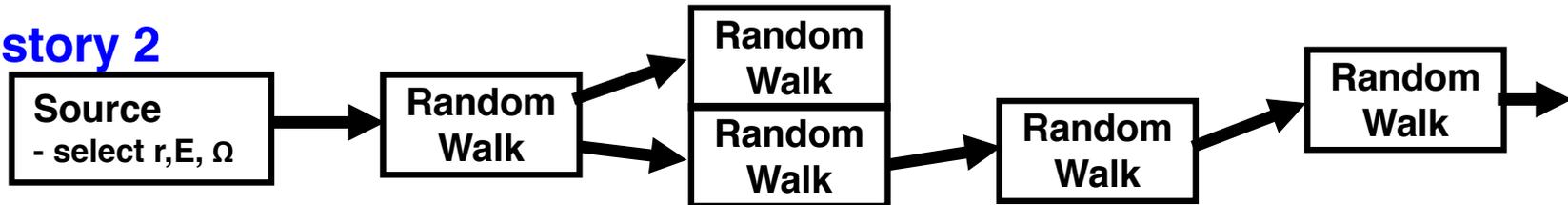


## Particle Histories

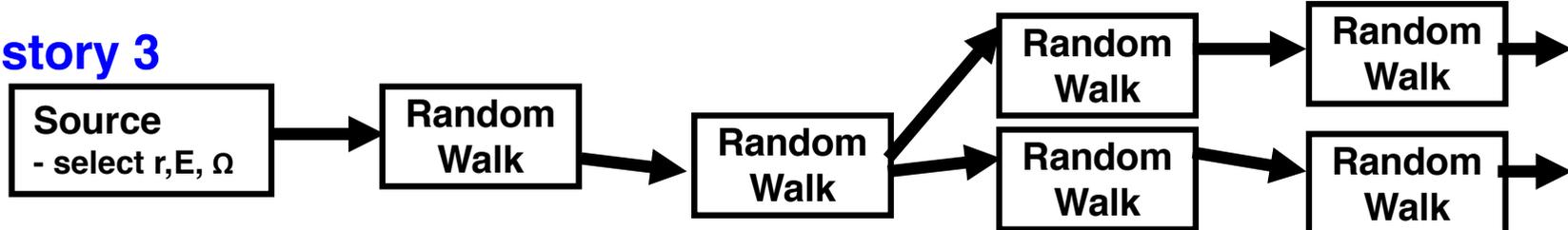
### History 1



### History 2



### History 3



# Monte Carlo Eigenvalue Problems

# Time-dependent Transport

$$\frac{1}{v} \frac{\partial \psi(\vec{r}, \mathbf{E}, \vec{\Omega}, t)}{\partial t} = \mathbf{Q} + [\mathbf{S} + \mathbf{M}] \cdot \psi - [\mathbf{L} + \mathbf{T}] \cdot \psi$$

- **Monte Carlo solution (over  $\Delta t$ , with fixed geometry & materials)**

- Simulate time-dependent transport for a neutron history
- If fission occurs, bank any secondary neutrons.
- When original particle is finished, simulate secondaries till done.
- Tallies for time bins, energy bins, cells, ...

- **At time  $t$ , the overall neutron level is** 
$$N(t) = \iiint_{\vec{r}, \mathbf{E}, \hat{\Omega}} \frac{\psi(\vec{r}, \mathbf{E}, \hat{\Omega}, t)}{v} d\vec{r} d\mathbf{E} d\hat{\Omega}$$

- **Alpha & T (reactor period,  $T = 1/\alpha$ ) can be defined by:**

$$N(t) = N_0 e^{\alpha t}$$

$$\alpha = \frac{d \ln N(t)}{dt} \approx \frac{\ln N(t) - \ln N_0}{t - t_0}$$

**This is the "dynamic alpha", NOT an eigenvalue !**

# Alpha Eigenvalue Equations

- For problems which are separable in space & time, it may be advantageous to solve a **static eigenvalue problem**, rather than a fully time-dependent problem
- **Assume:**
  1. Fixed geometry & materials
  2. No external source:  $Q(\mathbf{r}, \mathbf{E}, \Omega, t) = 0$
  3. **Separability:**  $\Psi(\mathbf{r}, \mathbf{E}, \Omega, t) = \Psi_\alpha(\mathbf{r}, \mathbf{E}, \Omega) e^{\alpha t}$ ,
- Substituting  $\Psi$  into the time-dependent transport equation yields

$$\left[ \mathbf{L} + \mathbf{T} + \frac{\alpha}{\mathbf{v}} \right] \Psi_\alpha(\vec{\mathbf{r}}, \mathbf{E}, \vec{\Omega}) = [\mathbf{S} + \mathbf{M}] \Psi_\alpha$$

- This is a static equation, an eigenvalue problem for  $\alpha$  and  $\Psi_\alpha$  without time-dependence
- $\alpha$  is often called the time-eigenvalue or time-absorption
- $\alpha$ -eigenvalue problems can be solved by Monte Carlo methods

# $K_{\text{eff}}$ Eigenvalue Equation

- For problems with fission multiplication, another approach is to create a **static eigenvalue problem** from the time-dependent transport equation (the asymptotic or steady-state solution)
- Introduce  $K_{\text{eff}}$ , a scaling factor on the multiplication ( $\nu$ )
- **Assume:**
  1. Fixed geometry & materials
  2. No external source:  $Q(\mathbf{r}, \mathbf{E}, \Omega, t) = 0$
  3.  $\partial\psi/\partial t = 0$ :  $\nu \Rightarrow \nu / K_{\text{eff}}$
- Setting  $\partial\psi/\partial t = 0$  and introducing the  $K_{\text{eff}}$  eigenvalue gives

$$[\mathbf{L} + \mathbf{T}] \Psi_k(\vec{\mathbf{r}}, \mathbf{E}, \vec{\Omega}) = \left[ \mathbf{S} + \frac{1}{K_{\text{eff}}} \mathbf{M} \right] \Psi_k$$

- Steady-state equation, a static eigenvalue problem for  $K_{\text{eff}}$  and  $\psi_k$
- $K_{\text{eff}}$  = effective multiplication factor
- Critical:  $K=1$ , subcritical:  $k < 1$ , supercritical:  $k > 1$
- $K_{\text{eff}}$  &  $\psi_k$  should never be used to model time-dependent problems.

# Comments on $K_{\text{eff}}$ and $\alpha$ Equations

---

- Criticality**

Supercritical:  $\alpha > 0$  or  $K_{\text{eff}} > 1$

Critical:  $\alpha = 0$  or  $K_{\text{eff}} = 1$

Subcritical:  $\alpha < 0$  or  $K_{\text{eff}} < 1$

- $K_{\text{eff}}$  vs.  $\alpha$  eigenvalue equations**

- $\Psi_k(r, E, \Omega) \neq \Psi_\alpha(r, E, \Omega)$ , except for a critical system

- $\alpha$  eigenvalue &  $\Psi_\alpha$  eigenfunction used for time-dependent problems

- $K_{\text{eff}}$  eigenvalue &  $\Psi_k$  eigenfunction used for reactor design & analysis

- Although  $\alpha = (K_{\text{eff}} - 1) / \lambda$ , where  $\lambda = \text{lifetime}$ ,  
there is no direct relationship between  $\Psi_k(r, E, \Omega)$  and  $\Psi_\alpha(r, E, \Omega)$

- $K_{\text{eff}}$  eigenvalue problems can be solved directly using Monte Carlo**

- $\alpha$  eigenvalue problems are solved by Monte Carlo indirectly using a series of  $K_{\text{eff}}$  calculations**

# Comments on $K_{\text{eff}}$ and $\alpha$ Equations

---

K equation  $[L + T] \Psi_k = [S + 1/k M] \Psi_k$

$\alpha$  equation  $[L + T + \alpha/v] \Psi_\alpha = [S + M] \Psi_\alpha$

- The factor  $1/k$  changes the relative level of the fission source
- The factor  $\alpha/v$  changes the absorption & neutron spectrum
  - For  $\alpha > 0$ , more absorption at low E  $\rightarrow$  harder spectrum
  - Double-density Godiva, average neutron energy causing fission:
    - k calculation: 1.30 MeV
    - $\alpha$  calculation: 1.68 MeV
- For **separable** problems,  $\Psi(r, E, \Omega, t) = \Psi_\alpha(r, E, \Omega) e^{\alpha t}$
- No similar equation for k, since not used for time-dependence

# K-eigenvalue equation

---

$$(\mathbf{L} + \mathbf{T})\Psi = \mathbf{S}\Psi + \frac{1}{K_{\text{eff}}}\mathbf{M}\Psi$$

where

$\mathbf{L}$  = leakage operator

$\mathbf{S}$  = scatter-in operator

$\mathbf{T}$  = collision operator

$\mathbf{M}$  = fission multiplication

operator

- Rearrange

$$(\mathbf{L} + \mathbf{T} - \mathbf{S})\Psi = \frac{1}{K_{\text{eff}}}\mathbf{M}\Psi$$

$$\Psi = \frac{1}{K_{\text{eff}}} \cdot (\mathbf{L} + \mathbf{T} - \mathbf{S})^{-1}\mathbf{M}\Psi$$

$$\Psi = \frac{1}{K_{\text{eff}}} \cdot \mathbf{F}\Psi$$

→ This eigenvalue equation will be solved by power iteration

$$\Psi^{(n+1)} = \frac{1}{K_{\text{eff}}^{(n)}} \cdot \mathbf{F}\Psi^{(n)}, \quad n = 0, 1, 2, \dots \quad \text{iteration}$$

# Power Iteration

## Diffusion Theory or Discrete-ordinates Transport

Initial guess for  $K_{\text{eff}}$  and  $\Psi$

$$K_{\text{eff}}^{(0)}, \Psi^{(0)}$$

**Outer iteration –**

Repeat until  $K_{\text{eff}}^{(n+1)}$  &  $\Psi^{(n+1)}$  converge

Solve for  $\Psi^{(n+1)}$

$$(\mathbf{L} + \mathbf{T} - \mathbf{S})\Psi^{(n+1)} = \frac{1}{K_{\text{eff}}^{(n)}} \mathbf{M}\Psi^{(n)}$$

**Inner iterations** - sweep over space  
or space/angle to solve for  $\Psi^{(n+1)}$

Update  $K_{\text{eff}}^{(n+1)}$

$$K_{\text{eff}}^{(n+1)} = K_{\text{eff}}^{(n)} \cdot \frac{1 \cdot \mathbf{M}\Psi^{(n+1)}}{1 \cdot \mathbf{M}\Psi^{(n)}}$$

**Done. Print results**

## Monte Carlo

Initial guess for  $K_{\text{eff}}$  and  $\Psi$

$$K_{\text{eff}}^{(0)}, \Psi^{(0)}$$

**Outer iteration –**

Repeat until  $K_{\text{eff}}^{(n+1)}$  &  $\Psi^{(n+1)}$  converge

Solve for  $\Psi^{(n+1)}$

$$(\mathbf{L} + \mathbf{T} - \mathbf{S})\Psi^{(n+1)} = \frac{1}{K_{\text{eff}}^{(n)}} \mathbf{M}\Psi^{(n)}$$

**Follow particle histories to solve for  $\Psi^{(n+1)}$**

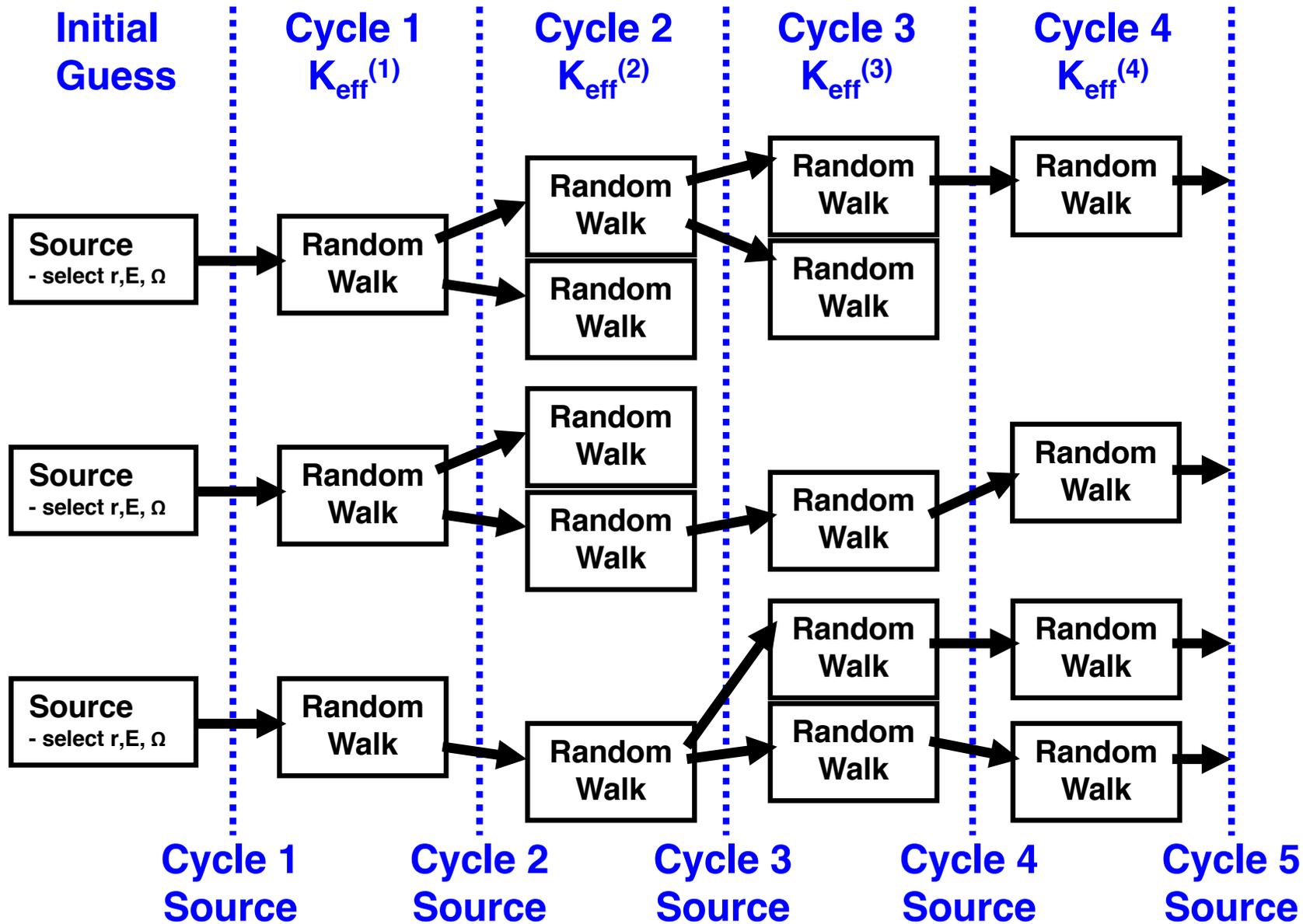
**During histories, save fission sites to use  
for source in next iteration**

**During histories, make tallies for  $K_{\text{eff}}^{(n+1)}$**

**Done, clear tallies.**

**Continue iterating, accumulate tallies**

# Monte Carlo Eigenvalue calculation



**Iterate (cycle) until converged, then more to accumulate tallies**

# $\alpha$ -Eigenvalue Calculations (Alpha search)

- **Eigenvalue equation with both  $K_{\text{eff}}$  &  $\alpha$** 
  - $\alpha$  is a fixed number, not a variable or eigenvalue
  - Find the k-eigenvalue as function of  $\alpha$ ,  $K(\alpha)$

$$\left[ \mathbf{L} + \mathbf{T} + \frac{\alpha}{v} \right] \Psi_{\alpha}(\vec{r}, \mathbf{E}, \vec{\Omega}) = \left[ \mathbf{S} + \frac{1}{K_{\text{eff}}} \mathbf{M} \right] \Psi_{\alpha}$$

- Note: If  $\alpha < 0$ 
  - Real absorption plus time absorption could be negative
  - Move  $\alpha/v$  to right side to prevent negative absorption,
  - $-\alpha/v$  term on right side is treated as a delta-function scatter
- Select a fixed value for  $\alpha$
- Solve the K-eigenvalue equations, with fixed time-absorption  $\alpha/v$
- Select a different  $\alpha$  and solve for a new  $K_{\text{eff}}$
- Repeat, searching for value of  $\alpha$  which results in  $K_{\text{eff}} = 1$

# K- and $\alpha$ -Eigenvalue Calculations

---

- **K-eigenvalue solution**

- **Loop for Power Iteration for K**

- Loop over neutrons in cycle
    - neutron history
    - ...
  - ...

Monte Carlo

- **$\alpha$ -eigenvalue solution**

- **Loop for  $\alpha$  search iterations**

- **Loop for Power Iteration for K**
  - Loop over neutrons in cycle
      - neutron history
      - ...
    - ...

Monte Carlo

→ Find  $K(\alpha)$ , then search for  $\alpha$  that gives  $K(\alpha)=1$

**Questions ?**



**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Adjoins & Green's Functions**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Outline

---

- **Introduction**
  - Forward & Adjoint Transport Equations
  - Superposition Principle
  - Green's functions & transport
- **Forward & Adjoint LBTE**
  - Integral equation for the neutron source
  - Integral equation for the adjoint source
  - Comments on forward vs adjoint
  - Relationship between forward & adjoint
- **Green's Function Approach**
  - Forward & adjoint Green's functions
  - K-eigenvalue form
  - Reciprocity
  - Discussion

---

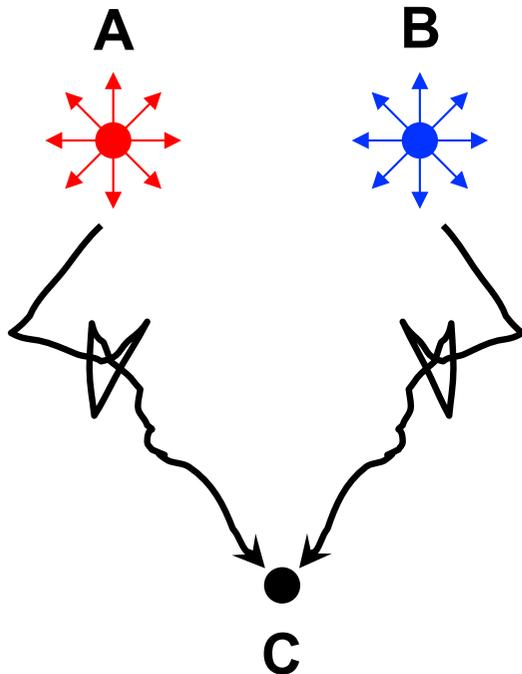
# Introduction

# Introduction – Forward & Adjoint Transport

---

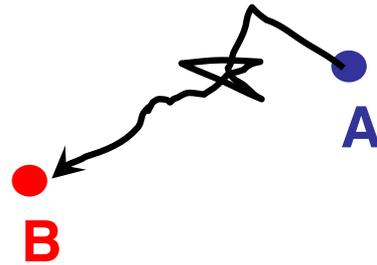
- Given a source, the **forward** LBTE gives the response at all points in the problem phase space
  - Forward LBTE describes **where the particles will go**
- Given a response, the **adjoint** LBTE gives the source at all points in the problem phase space that would produce the response
  - Adjoint LBTE describes **where the particles came from**
  - The adjoint LBTE essentially follows particles backwards (in  $\Omega, E, t$ ) from the response to the source
  - For fixed-source problems, the response is a **particular tally**
  - For eigenvalue problems, the response is the **forward fundamental mode solution** (ie, the fission neutron distribution)
  - The adjoint solution is often called the **importance**

# Introduction – Superposition Principle



- Consider 2 sources, A & B, and one detector point C
  - Denote the flux response at point C by  $\phi_C$
  - If source **A is on** & source **B is off**,
    - Solve the LBTE to get the flux response at point C due to the source at point A,  $\phi_{A \rightarrow C}$
    - $\phi_C = \phi_{A \rightarrow C}$
  - If source **A is off** & source **B is on**,
    - Solve the LBTE to get the flux response at point C due to the source at point A,  $\phi_{B \rightarrow C}$
    - $\phi_C = \phi_{B \rightarrow C}$
  - If source **A is on** & source **B is on**,
    - $\phi_C = \phi_{A \rightarrow C} + \phi_{B \rightarrow C}$
- Linearity of LBTE permits adding the response from different sources to get the total source

# Introduction - Green's Functions & Transport Theory



$$S_B = S_A \cdot G(A \rightarrow B)$$

- $G(A \rightarrow B)$

- Green's function, "here-to-there" function
- Probability that source at point A produces source at point B

- **Transport theory - Peierl's equation for multiplying system**

$$S(\vec{r}) = \frac{1}{k_{\text{eff}}} \cdot \int_{\text{all } \vec{r}'} d\vec{r}' \cdot S(\vec{r}') \cdot G(\vec{r}' \rightarrow \vec{r})$$

- $G(r' \rightarrow r)$  gives the fission source at  $r$  (in a single generation) due to a fission neutron born at  $r'$
- This use of a Green's function is considered "obvious", but it is based on rigorous math (ie, integral operator theory)

---

# Forward & Adjoint LBTE

# Time-independent, Including Fission

- Time-independent **forward LBTE**

$$\begin{aligned} & \hat{\Omega} \cdot \nabla \Psi(\vec{r}, E, \hat{\Omega}) \\ & + \Sigma_T(\vec{r}, E) \Psi(\vec{r}, E, \hat{\Omega}) \\ & - \iint dE' d\hat{\Omega}' \Sigma_S(\vec{r}, E' \rightarrow E, \hat{\Omega} \bullet \hat{\Omega}') \Psi(\vec{r}, E', \hat{\Omega}') \\ & = S(\vec{r}, E, \hat{\Omega}) \end{aligned}$$

Short form:  $L\Psi(\vec{r}, E, \hat{\Omega}) = S(\vec{r}, E, \hat{\Omega})$

For fixed-source problem,

S is an internal or volume source

For eigenvalue problem,

$$S(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \frac{\chi(E)}{4\pi} \iint dE' d\hat{\Omega}' v\Sigma_F(\vec{r}, E') \Psi(\vec{r}, E', \hat{\Omega}')$$

- Time-independent **adjoint LBTE**

$$\begin{aligned} & -\hat{\Omega} \cdot \nabla \Psi^\dagger(\vec{r}, E, \hat{\Omega}) \\ & + \Sigma_T(\vec{r}, E) \Psi^\dagger(\vec{r}, E, \hat{\Omega}) \\ & - \iint dE' d\hat{\Omega}' \Sigma_S(\vec{r}, E \rightarrow E', -\hat{\Omega} \bullet \hat{\Omega}') \Psi^\dagger(\vec{r}, E', \hat{\Omega}') \\ & = S^\dagger(\vec{r}, E, \hat{\Omega}) \end{aligned}$$

Short form:  $L^\dagger \Psi^\dagger(\vec{r}, E, \hat{\Omega}) = S^\dagger(\vec{r}, E, \hat{\Omega})$

For fixed-source problem,

S<sup>†</sup> is a specific tally response

For eigenvalue problem,

$$S^\dagger(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot v\Sigma_F(\vec{r}, E) \iint dE' d\hat{\Omega}' \cdot \frac{\chi(E')}{4\pi} \cdot \Psi^\dagger(\vec{r}, E', \hat{\Omega}')$$

Reverse  $\hat{\Omega}$  to  $-\hat{\Omega}$   
 Interchange  $E'$  and  $E$   
 Interchange  $v\Sigma_F$  and  $\chi/4\pi$

# Adjoint = Importance

---

- **Why are adjoint solutions needed?**

- In quantum theory, operators that produce measurable results are Hermitian (or self-adjoint). Complete sets of orthogonal eigenfunctions exist.
- In 1-speed transport theory or 1-group diffusion theory
  - The operators are self-adjoint (kernels are symmetric)
  - The LBTE has a complete set of orthogonal eigenfunctions
  - Forward & adjoint eigenfunctions are the same
- For energy-dependent transport & multigroup diffusion
  - The operators are not self-adjoint (kernels are not symmetric)
  - Eigenfunctions for the forward LBTE are not orthogonal & are different from the adjoint eigenfunctions
  - However, the forward & adjoint eigenfunctions are biorthogonal,  
 $\int \psi_p^\dagger \psi_q = 0$  if  $p \neq q$
- **First-order perturbation theory:**  $\Delta \rho = \langle \psi^\dagger \Delta \mathbf{x} \psi \rangle / \langle \psi^\dagger \mathbf{F} \psi \rangle$ 
  - Change in parameter weighted by importance
- **Reactor kinetics:**

$$\Lambda_{\text{eff}} = \langle \psi^\dagger \mathbf{1}/v \psi \rangle / \langle \psi^\dagger \mathbf{F} \psi \rangle$$

$$\beta_{\text{eff}} = \langle \psi^\dagger \mathbf{B} \psi \rangle / \langle \psi^\dagger \mathbf{F} \psi \rangle$$
  - Importance weighting

# MC Simulation

---

- **MC simulation gives the solution to the forward LBTE**
- **For some special cases, the MC simulation can be run backwards**
  - 1-speed problems
  - Multigroup problems (transpose the scattering matrix)
- **For general, energy-dependent problems, the MC simulation cannot be run backwards to get the adjoint LBTE solution**
  - **Some reactions can't be sampled backwards**
    - scattering with correlated  $E', \mu$  exit parameters
    - some inelastic scatters
    - etc.

# Green's Functions

# Forward Equations

---

- Define the net loss operator in the LBTE as

$$L \cdot \Psi(\vec{r}, E, \hat{\Omega}) = \hat{\Omega} \cdot \nabla \Psi(\vec{r}, E, \hat{\Omega}) + \Sigma_T(\vec{r}, E) \Psi(\vec{r}, E, \hat{\Omega}) \\ - \iint dE' d\hat{\Omega}' \Sigma_S(\vec{r}, E' \rightarrow E, \hat{\Omega}' \cdot \hat{\Omega}) \Psi(\vec{r}, E', \hat{\Omega}')$$

- The Green's function is the solution to the LBTE for a point source

$$L \cdot G(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega}) = \delta(\vec{r} - \vec{r}_0) \cdot \delta(E - E_0) \cdot \delta(\hat{\Omega} - \hat{\Omega}_0),$$

– By convention,  $G(r_0, E_0, \Omega_0 \rightarrow r, E, \Omega)$  is used, rather than  $G(r, E, \Omega)$

- The LBTE solution for an arbitrary source can then be written as

$$\Psi(\vec{r}, E, \hat{\Omega}) = \iiint d\vec{r}_0 dE_0 d\hat{\Omega}_0 S(\vec{r}_0, E_0, \hat{\Omega}_0) \cdot G(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

# Backward Equations

---

- Define the net loss operator in the LBTE as

$$L^\dagger \cdot \Psi^\dagger(\vec{r}, E, \hat{\Omega}) = -\hat{\Omega} \cdot \nabla \Psi^\dagger(\vec{r}, E, \hat{\Omega}) + \Sigma_T(\vec{r}, E) \Psi^\dagger(\vec{r}, E, \hat{\Omega}) \\ - \iint dE' d\hat{\Omega}' \Sigma_S(\vec{r}, E \rightarrow E', -\hat{\Omega}' \cdot \hat{\Omega}) \Psi^\dagger(\vec{r}, E', \hat{\Omega}')$$

- The Green's function is the solution to the LBTE for a point source

$$L^\dagger \cdot G^\dagger(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega}) = \delta(\vec{r} - \vec{r}_0) \cdot \delta(E - E_0) \cdot \delta(\hat{\Omega} - \hat{\Omega}_0)$$

– By convention,  $G^\dagger(r_0, E_0, \Omega_0 \rightarrow r, E, \Omega)$  is used, rather than  $G^\dagger(r, E, \Omega)$

- The LBTE solution for an arbitrary source can then be written as

$$\Psi^\dagger(\vec{r}, E, \hat{\Omega}) = \iiint d\vec{r}_0 dE_0 d\hat{\Omega}_0 S^\dagger(\vec{r}_0, E_0, \hat{\Omega}_0) \cdot G^\dagger(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

# k-eigenvalue Equations

---

- **Transport equation, k-eigenvalue form**

$$L \cdot \Psi(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \frac{\chi(E)}{4\pi} \cdot S(\vec{r}) \quad S(\vec{r}) = \iint dE' d\hat{\Omega}' v \Sigma_F(\vec{r}, E') \Psi(\vec{r}, E', \hat{\Omega}')$$

**Solution using Green's function**

$$\Psi(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \iiint d\vec{r}_0 dE_0 d\hat{\Omega}_0 \frac{\chi(E_0)}{4\pi} \cdot S(\vec{r}_0) \cdot G(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

- **Adjoint transport equation, k-eigenvalue form**

$$L^\dagger \cdot \Psi^\dagger(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot v \Sigma_F(\vec{r}, E) \cdot S^\dagger(\vec{r}) \quad S^\dagger(\vec{r}) = \iint dE' d\hat{\Omega}' \cdot \frac{\chi(E')}{4\pi} \cdot \Psi^\dagger(\vec{r}, E', \hat{\Omega}')$$

**Solution using Green's function**

$$\Psi^\dagger(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \iiint d\vec{r}_0 dE_0 d\hat{\Omega}_0 \cdot v \Sigma_F(\vec{r}_0, E_0) \cdot S^\dagger(\vec{r}_0) \cdot G^\dagger(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

# Reciprocity

---

- **G and G<sup>†</sup> are not symmetric, can't reverse  $r_0, E_0, \hat{\Omega}_0$  and  $r, E, \hat{\Omega}$**
- **Reciprocity for direct & adjoint Green's function**

$$G^\dagger( \vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega} ) = G( \vec{r}, E, \hat{\Omega} \rightarrow \vec{r}_0, E_0, \hat{\Omega}_0 )$$

- **Because of irreversible energy dependence, neither G nor G<sup>†</sup> is symmetric in initial and final arguments.**

- **Apply reciprocity to the adjoint Green's function solution**

$$\Psi^\dagger(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \iiint d\vec{r}_0 dE_0 d\hat{\Omega}_0 \cdot v \Sigma_F(\vec{r}_0, E_0) \cdot S^\dagger(\vec{r}_0) \cdot G(\vec{r}, E, \hat{\Omega} \rightarrow \vec{r}_0, E_0, \hat{\Omega}_0)$$

- **Compare with forward**

$$\Psi(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \iiint d\vec{r}_0 dE_0 d\hat{\Omega}_0 \frac{\chi(E_0)}{4\pi} \cdot S(\vec{r}_0) \cdot G(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

# Discussion

---

- **Forward & backward solutions**

$$\Psi(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \iiint d\vec{r}_0 dE_0 d\hat{\Omega}_0 \frac{\chi(E_0)}{4\pi} \cdot S(\vec{r}_0) \cdot G(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

$$\Psi^\dagger(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \iiint d\vec{r}_0 dE_0 d\hat{\Omega}_0 \cdot v\Sigma_F(\vec{r}_0, E_0) \cdot S^\dagger(\vec{r}_0) \cdot G(\vec{r}, E, \hat{\Omega} \rightarrow \vec{r}_0, E_0, \hat{\Omega}_0)$$

- **Why does this matter?**

**MC simulation computes  $G(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$  directly**

**Can pick starting points  $\vec{r}_0, E_0, \hat{\Omega}_0$ , then record tallies at  $\vec{r}, E, \hat{\Omega}$  with appropriate weighting functions**

# Discussion

---

- **Why does this matter ?**
  - **Green's function approach enables the use of the very rich mathematical tools from linear operator theory**
  - **Linear operator theory can be used to examine the existence & completeness of eigenfunction expansions**
  - **Green's function approach enables development of different Monte Carlo approaches**
  - **Next lecture on the fission matrix method is an example**
  - **Variance reduction methods attempt to influence the endpoints  $r, E, \Omega$**







**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Fission Matrix Method for Monte Carlo Criticality Problems**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Abstract

---

## **Fission Matrix Method for Monte Carlo Criticality Problems**

**Forrest Brown**

The theory underlying the fission matrix method is derived using a rigorous Green's function approach. The method is then used to investigate fundamental properties of the transport equation for a continuous-energy physics treatment. We provide evidence that an infinite set of discrete real eigenvalues and eigenfunctions exist for the continuous-energy problem, and that the eigenvalue spectrum converges smoothly as the spatial mesh for the fission matrix is refined.

We also derive equations for the adjoint solution. We show that if the mesh is sufficiently refined so that both forward and adjoint solutions are valid, then the adjoint fission matrix is identical to the transpose of the forward matrix. While the energy-dependent transport equation is strictly biorthogonal, we provide surprising results that the forward modes are very nearly self-adjoint for a variety of continuous-energy problems.

# Outline

---

- **Introduction**
  - Higher eigenmodes
  - Green's functions & transport
  - Motivation
- **Theoretical Basis of the Fission Matrix**
  - Integral equation for the neutron source
  - Integral equation for the adjoint source
  - Comments of forward vs adjoint
- **Forward & Adjoint Fission Matrix Equations**
  - Forward fission matrix equations
  - Adjoint fission matrix equations
  - Relationship between forward & adjoint
- **Fission Matrix Eigenmodes & Eigenvalue Spectrum**
  - Higher mode analysis
  - Spectrum convergence with mesh refinement
  - Real vs Complex eigenvalues
  - Near-orthogonality of eigenfunctions
- **Conclusions & Future Work**

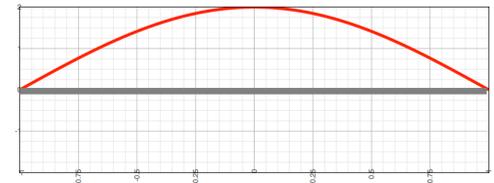
# Introduction

# Higher Eigenmodes

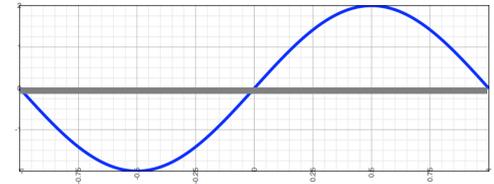


## Vibrating strings:

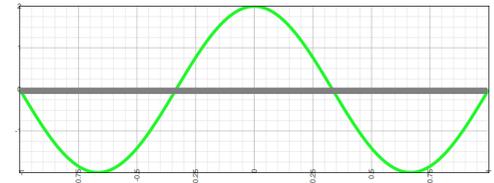
- Higher modes add "tone", but die away quickly
- **Fundamental mode persists**
- Feedback, instability, nonlinear effects, ..., may excite higher modes



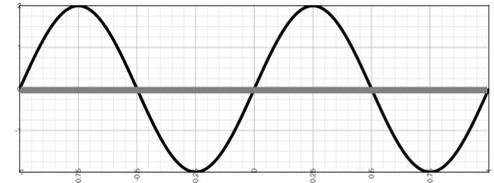
0



1



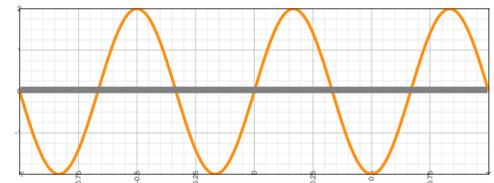
2



3



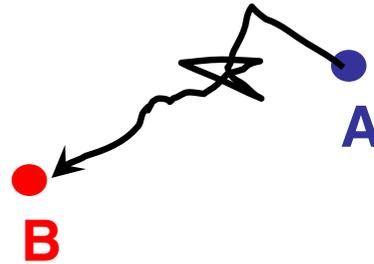
4



5

etc.

# Introduction - Green's Functions & Transport Theory



$$S_B = S_A \cdot F(A \rightarrow B)$$

- $F(A \rightarrow B)$ 
  - Green's function, "here-to-there" function
  - Probability that source at point A produces source at point B
- Transport theory - Peierl's equation for multiplying system

$$S(\vec{r}) = \frac{1}{k_{\text{eff}}} \cdot \int_{\text{all } \vec{r}'} d\vec{r}' \cdot S(\vec{r}') \cdot F(\vec{r}' \rightarrow \vec{r})$$

- Discretize space into blocks, or mesh regions
- Compute  $F(r' \rightarrow r)$  with Monte Carlo
- Solve matrix eigenvalue problem for sources:

$$\vec{S} = \frac{1}{k_{\text{eff}}} \cdot \bar{F} \cdot \vec{S}$$

- Can also solve for higher modes

# Introduction – Who Cares?

---

- **Knowledge of fundamental & all higher modes**
  - “Crown Jewels” of analysis – explains everything
- **Reactor theory & mathematical foundations**
  - Existence of higher modes
  - Eigenvalue spectrum – discrete ? real ?
  - Forward & adjoint modes
  - Assessment of spatial refinement
- **Fundamental reactor physics analysis**
  - Higher modes for stability analysis of Xenon & void oscillations
  - Slow-transient analysis
  - Startup, probability of initiation
- **Source convergence testing & acceleration**
  - May provide robust, reliable, automated convergence test
  - Acceleration of source convergence

# Theoretical Basis of the Fission Matrix

# Integral Equation for the Neutron Source (1)

- Transport equation, k-eigenvalue form

$$M \cdot \Psi(\vec{r}, E, \hat{\Omega}) = \frac{1}{k} \cdot \frac{\chi(E)}{4\pi} \cdot S(\vec{r})$$

**M = net loss operator**

$$M \cdot \Psi(\vec{r}, E, \hat{\Omega}) = \hat{\Omega} \cdot \nabla \Psi(\vec{r}, E, \hat{\Omega}) + \Sigma_T(\vec{r}, E) \Psi(\vec{r}, E, \hat{\Omega}) \\ - \iint dE' d\hat{\Omega}' \Sigma_S(\vec{r}, E' \rightarrow E, \hat{\Omega}' \rightarrow \hat{\Omega}) \Psi(\vec{r}, E', \hat{\Omega}')$$

**S(r) = fission neutron source**

$$S(\vec{r}) = \iint dE' d\hat{\Omega}' v \Sigma_F(\vec{r}, E') \Psi(\vec{r}, E', \hat{\Omega}')$$

**$\chi(E)$  = emission spectrum,**  
**following analysis is same if replaced by**

$$\chi(E, \vec{r}) = \frac{\iint dE' d\hat{\Omega}' \chi(E' \rightarrow E) v \Sigma_F(\vec{r}, E') \Psi(\vec{r}, E', \hat{\Omega}')} {\iint dE' d\hat{\Omega}' v \Sigma_F(\vec{r}, E') \Psi(\vec{r}, E', \hat{\Omega}')}$$

## Integral Equation for the Neutron Source (2)

- Define Green's function & integral transport equation

$$M \cdot G(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega}) = \delta(\vec{r} - \vec{r}_0) \cdot \delta(E - E_0) \cdot \delta(\hat{\Omega} - \hat{\Omega}_0),$$

$$\Psi(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \iiint d\vec{r}_0 dE_0 d\hat{\Omega}_0 \frac{\chi(E_0)}{4\pi} \cdot S(\vec{r}_0) \cdot G(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

- Multiply by  $v\Sigma_F(r, E)$ , integrate over  $E, \Omega$
- Define energy-angle averaged Source & Green's function

$$H(\vec{r}_0 \rightarrow \vec{r}) = \int \iiint dE d\hat{\Omega} dE_0 d\hat{\Omega}_0 \cdot v\Sigma_F(\vec{r}, E) \cdot \frac{\chi(E_0)}{4\pi} \cdot G(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

$$S(\vec{r}) = \iint dE' d\hat{\Omega}' \cdot v\Sigma_F(\vec{r}, E') \cdot \Psi(\vec{r}, E', \hat{\Omega}')$$

$$S(\vec{r}) = \frac{1}{K} \int d\vec{r}_0 \cdot S(\vec{r}_0) \cdot H(\vec{r}_0 \rightarrow \vec{r})$$

**$H(r_0 \rightarrow r)$  can be tallied directly in MC simulation**

# Integral Equation for the Adjoint Neutron Source (1)

- Adjoint transport equation, k-eigenvalue form

$$M^\dagger \cdot \Psi^\dagger(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \frac{v \Sigma_F(\vec{r}, E)}{4\pi} \cdot S^\dagger(\vec{r})$$

$M^\dagger$  = adjoint to operator M

$$M^\dagger \cdot \Psi^\dagger(\vec{r}, E, \hat{\Omega}) = -\hat{\Omega} \cdot \nabla \Psi^\dagger(\vec{r}, E, \hat{\Omega}) + \Sigma_T(\vec{r}, E) \Psi^\dagger(\vec{r}, E, \hat{\Omega}) \\ - \iint dE' d\hat{\Omega}' \cdot \Sigma_S(\vec{r}, E \rightarrow E', \hat{\Omega} \rightarrow \hat{\Omega}') \cdot \Psi^\dagger(\vec{r}, E', \hat{\Omega}')$$

$S^\dagger(\vec{r})$  = adjoint fission neutron source

$$S^\dagger(\vec{r}) = \iint dE' d\hat{\Omega}' \cdot \frac{\chi(E')}{4\pi} \cdot \Psi^\dagger(\vec{r}, E', \hat{\Omega}')$$

Bell & Glasstone & others have shown that forward & adjoint K eigenvalues are the same,  $K^\dagger = K$ , so will just use K in the following analysis.

## Integral Equation for the Adjoint Neutron Source (2)

- **Adjoint Green's function & integral transport equation**

$$M^\dagger \cdot G^\dagger(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega}) = \delta(\vec{r} - \vec{r}_0) \cdot \delta(E - E_0) \cdot \delta(\hat{\Omega} - \hat{\Omega}_0)$$

$$\Psi^\dagger(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \iiint d\vec{r}_0 dE_0 d\hat{\Omega}_0 \cdot v\Sigma_F(\vec{r}_0, E_0) \cdot S^\dagger(\vec{r}_0) \cdot G^\dagger(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

- **Multiply by  $\chi(E)$ , integrate over  $E, \Omega$**
- **Define energy-angle averaged adjoint Source & Green's function**

$$H^\dagger(\vec{r}_0 \rightarrow \vec{r}) = \int \iiint dE d\hat{\Omega} dE_0 d\hat{\Omega}_0 \cdot \frac{\chi(E)}{4\pi} \cdot v\Sigma_F(\vec{r}_0, E_0) \cdot G^\dagger(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

$$S^\dagger(\vec{r}) = \iint dE' d\hat{\Omega}' \cdot \frac{\chi(E')}{4\pi} \cdot \Psi^\dagger(\vec{r}', E', \hat{\Omega}')$$

$$S^\dagger(\vec{r}) = \frac{1}{K} \int d\vec{r}_0 \cdot S^\dagger(\vec{r}_0) \cdot H^\dagger(\vec{r}_0 \rightarrow \vec{r})$$

# Forward & Adjoint Integral Equations for Source

- **Reciprocity for direct & adjoint Green's function**

$$G^\dagger(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega}) = G(\vec{r}, E, \hat{\Omega} \rightarrow \vec{r}_0, E_0, \hat{\Omega}_0)$$

Because of irreversible energy dependence, neither  $G$  nor  $G^\dagger$  is symmetric in initial and final arguments. Same is true for  $H$  and  $H^\dagger$

$$H^\dagger(\vec{r}_0 \rightarrow \vec{r}) = H(\vec{r} \rightarrow \vec{r}_0)$$

$$H(\vec{r}_0 \rightarrow \vec{r}) \neq H(\vec{r} \rightarrow \vec{r}_0),$$

$$H^\dagger(\vec{r}_0 \rightarrow \vec{r}) \neq H^\dagger(\vec{r} \rightarrow \vec{r}_0)$$

- **Using reciprocity, comparing  $H$  and  $H^\dagger$  gives**

$$S(\vec{r}) = \frac{1}{K} \int d\vec{r}_0 \cdot S(\vec{r}_0) \cdot H(\vec{r}_0 \rightarrow \vec{r})$$

$$S^\dagger(\vec{r}) = \frac{1}{K} \int d\vec{r}_0 \cdot S^\dagger(\vec{r}_0) \cdot H(\vec{r} \rightarrow \vec{r}_0)$$

- **$S$  and  $S^\dagger$  are bi-orthogonal**

$$(K_p - K_q) \cdot \int d\vec{r} \cdot S_p(\vec{r}) \cdot S_q^\dagger(\vec{r}) = 0$$

# K-eigenvalue Form of Transport Equation

---

- **Structure & properties**

$$M \cdot \Psi(\vec{r}, E, \hat{\Omega}) = \frac{1}{k} \cdot \frac{\chi(E)}{4\pi} \cdot S(\vec{r})$$

- 60+ years ago:

A single, non-negative, real, fundamental eigenfunction & eigenvalue exist

- 50+ years ago:

For 1-speed or 1-group: A complete set of self-adjoint, real eigenfunctions & discrete eigenvalues exists

- Energy-dependent transport equation is bi-orthogonal, forward & adjoint modes are orthogonal

- Nothing else proven, always assumed that higher-mode solutions exist

- **In the present work based on the Fission Matrix:**

- We provide evidence that higher modes exist, are real, have discrete eigenvalues, and are very nearly self-adjoint (for reactor-like problems)

- Approach is similar to Birkhoff's original proof for fundamental mode

- This has never been done before using continuous-energy Monte Carlo

# Forward & Adjoint Fission Matrix Equations

# Forward Fission Matrix Equations (1)

---

- **Segment the physical problem into N disjoint spatial regions**
  - Initial regions ( $r_0$ ) for fission neutron source emission
  - Final regions ( $r$ ) for production of a next-generation fission neutron
- **Integrate the forward integral fission source equation over  $r_0$  &  $r$** 
  - Initial:  $r_0 \in V_J$ ,      Final:  $r \in V_I$

$$S_I = \frac{1}{K} \cdot \sum_{J=1}^N F_{I,J} \cdot S_J$$

$$F_{I,J} = \int_{\vec{r} \in V_I} d\vec{r} \int_{\vec{r}_0 \in V_J} d\vec{r}_0 \frac{S(\vec{r}_0)}{S_J} \cdot H(\vec{r}_0 \rightarrow \vec{r}) \quad S_J = \int_{\vec{r}' \in V_J} S(\vec{r}') d\vec{r}'$$

**Exact equations for integral source  $S_I$**

**$N = \#$  spatial regions,  $F = N \times N$  matrix, nonsymmetric**

## Forward Fission Matrix Equations (2)

---

- $F_{I,J}$  = next-generation fission neutrons produced in region I, for each average fission neutron starting in region J ( $J \rightarrow I$ )
- In the equation for F,
  - $S(r_0)/S_J$  is a local weighting function within region J
  - As  $V_J \rightarrow 0$ :
    - $S(r_0) \rightarrow S_J / V_J$
    - Discretization errors  $\rightarrow 0$
    - Can accumulate tallies of  $F_{I,J}$  even if not converged
- $F_{I,J}$  tallies:
  - Previous F-matrix work: tally during neutron random walks
  - Present F-matrix work: tally only point-to-point, using fission-bank in master proc (~free)
    - Eliminates excessive communications for parallel
    - Provides more consistency,  $F_{I,J}$  nonzero only in elements with actual sites
    - Analog-like treatment, better for preserving overall balance

# Adjoint Fission Matrix Equations

---

- **Segment the physical problem into N disjoint spatial regions**
  - Initial regions:  $r_0 \in V_J$ ,    Final regions:  $r \in V_I$
- **Integrate the adjoint integral fission source equation over  $r_0$  &  $r$**

$$S_I^\dagger = \frac{1}{K} \cdot \sum_{J=1}^N F_{I,J}^\dagger \cdot S_J^\dagger$$

$$F_{I,J}^\dagger = \int_{\vec{r} \in V_I} d\vec{r} \int_{\vec{r}_0 \in V_J} d\vec{r}_0 \frac{S^\dagger(\vec{r}_0)}{S_J^\dagger} \cdot H(\vec{r} \rightarrow \vec{r}_0) \quad S_J^\dagger = \int_{\vec{r}' \in V_J} S^\dagger(\vec{r}') d\vec{r}'$$

**Exact equations for adjoint integral source  $S_I^\dagger$**

# Relationship Between Forward & Adjoint Fission Matrix

- $F_{I,J}$  = next-generation fission neutrons produced in region I, for each average fission neutron starting in region J (J→I)
- Compare  $F_{I,J}$  &  $F_{J,I}^\dagger$ , interchange integration order for  $F_{J,I}^\dagger$

$$F_{I,J} = \int_{\vec{r} \in V_I} d\vec{r} \int_{\vec{r}_0 \in V_J} d\vec{r}_0 \cdot \frac{S(\vec{r}_0)}{S_J} \cdot H(\vec{r}_0 \rightarrow \vec{r})$$

Same form, but different spatial weighting functions

$$F_{J,I}^\dagger = \int_{\vec{r}_0 \in V_J} d\vec{r}_0 \int_{\vec{r} \in V_I} d\vec{r} \cdot \frac{S^\dagger(\vec{r})}{S_I} \cdot H(\vec{r}_0 \rightarrow \vec{r})$$

- If the spatial discretization is fine enough that

$$\frac{S(\vec{r}_0)}{S_J/V_J} \approx 1 \quad \text{for } \vec{r}_0 \in V_J \quad \text{and} \quad \frac{S^\dagger(\vec{r})}{S_I/V_I} \approx 1 \quad \text{for } \vec{r} \in V_I$$

then

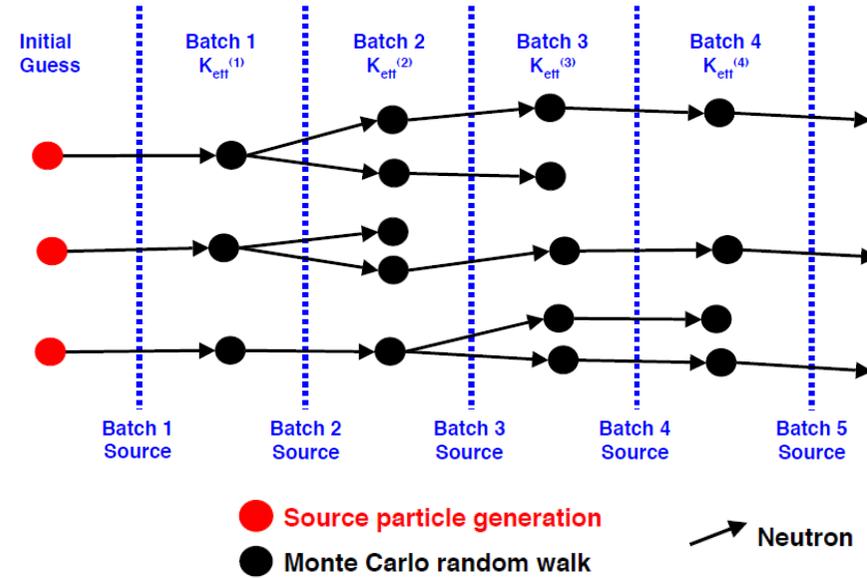
- Discretization errors from neglecting weights  $\rightarrow 0$
- Can accumulate tallies of  $F_{I,J}$  even if not converged
- For fine spatial mesh,  $F^\dagger = \text{transpose of } F$

$$\bar{F}^\dagger = \bar{F}^T$$

# Monte Carlo Estimation of Fission Matrix

## Monte Carlo K-effective Calculation

1. Start with fission source & k-eff guess
2. Repeat until converged:
  - Simulate neutrons in cycle
  - Save fission sites for next cycle
  - Calculate k-eff, renormalize source
3. Continue iterating & tally results



## For Fission Matrix calculation

During standard k-eff calculation, at the end of each cycle:

- Estimate  $F_{i,j}$  tallies from start & end points in fission bank (~ free)
- Accumulate  $F_{i,j}$  tallies, over all cycles (even inactive cycles)

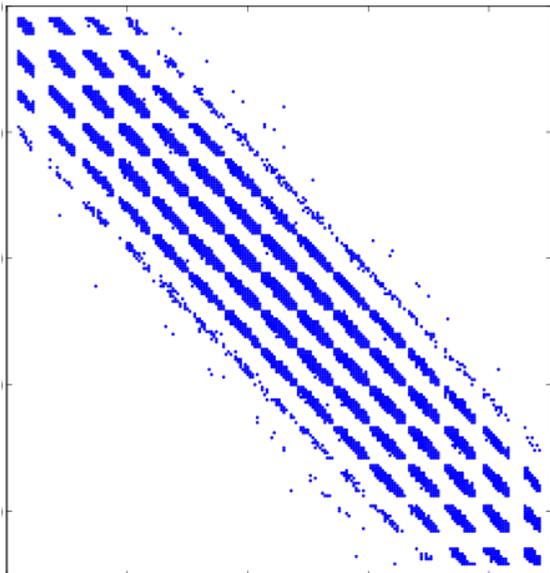
After Monte Carlo completed:

- Normalize  $F_{i,j}$  accumulators, divide by total sources in J regions
- Find eigenvalues/vectors of F matrix (power iteration, with deflation)

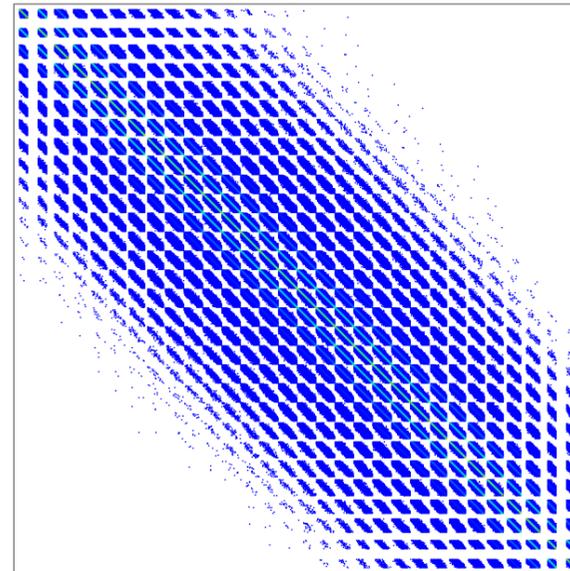
# Fission Matrix – Sparse Structure

- For a spatial mesh with  $N$  regions,  $F$  matrix is  $N \times N$ 
  - 100x100x100 mesh  $\rightarrow F$  is  $10^6 \times 10^6 \rightarrow 8$  TB memory
  - In the past, memory storage was always the major limitation for  $F$  matrix
- **Compressed row storage scheme**
  - Don't store near-zero elements, general sparsity
  - Reduced  $F$  matrix storage, no approximation
  - Can easily do 100x100x100 mesh on 8 GB Mac

2D PWR - 15x15x1 mesh, N=225



2D PWR - 30x30x1 mesh, N=900



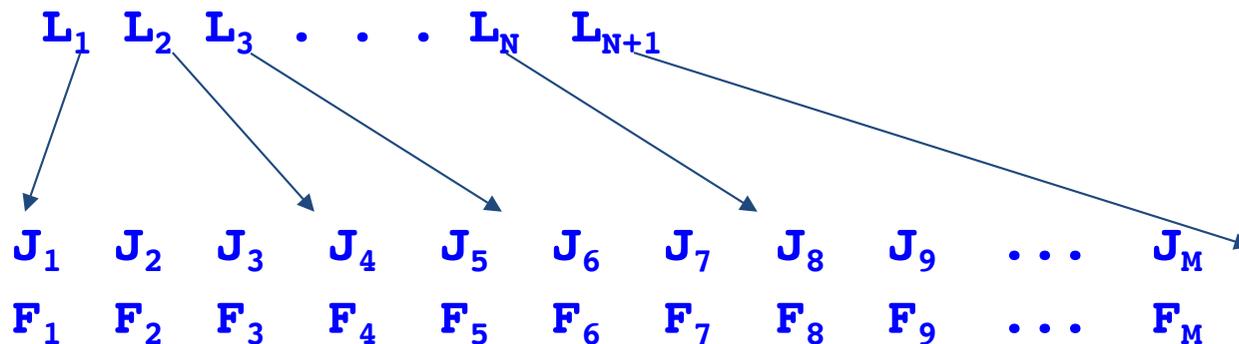
# Fission Matrix – Sparse Storage

## • Compressed Row Storage Scheme (CRS)

- General sparsity, no approximations or assumptions
- $N = N_x \times N_y \times N_z$  mesh cells
- $(i_S, j_S, k_S) \rightarrow (i_T, j_T, k_T) \rightarrow J \rightarrow I$ 

$$J = i_S + (j_S - 1)N_x + (k_S - 1)N_x N_y$$

$$I = i_T + (j_T - 1)N_x + (k_T - 1)N_x N_y$$
- Only the nonzero  $F(I, J)$  entries are stored.
- **MC tallies:** If element exists – add to it; if not – insert it
- $L(I)$  array entries point to the start of a list of  $J$  indices and corresponding nonzero  $F(I, J)$  tallies



- **Highly optimized tally coding, typically requires less than 1 second at the end of each batch in the Monte Carlo simulation.**

## Example – Sparse-Matrix \* Vector

---

```

! multiply a fmat matrix times a vector, return result in y vector
type(fission_matrix), intent(in)  :: fmat    ! sparse fission matrix
real(R8),             intent(in)  :: x(:)    ! vector in
real(R8),             intent(out) :: y(:)    ! vector out, result
integer(I8) :: k, i
real(R8)      :: t

!$OMP PARALLEL DO PRIVATE( t, k )                ← different thread for each row
do i = 1, fmat%n
  t = 0.0d+00
  do k = fmat%L(i), fmat%L(i+1)-1                ← k is location of J,R row data
    t = t + fmat%R(k) * x(fmat%J(k))
  enddo
  y(i) = t
enddo
!$OMP END PARALLEL DO

```

# **Fission Matrix Eigenmodes & Eigenvalue Spectrum**

# K-eigenvalue Form of Transport Equation

$$M \cdot \Psi(\vec{r}, E, \hat{\Omega}) = \frac{1}{K} \cdot \frac{\chi(E)}{4\pi} \cdot S(\vec{r})$$

- **Structure & properties**

- 60+ years ago:

A single, non-negative, real, fundamental eigenfunction & eigenvalue exist (Birkhoff)

- 50+ years ago:

For 1-speed or 1-group: A complete set of self-adjoint, real eigenfunctions & discrete eigenvalues exists (Lehner & Wing, Sahni)

- Energy-dependent transport equation is bi-orthogonal, forward & adjoint modes are orthogonal

- Nothing else proven, always assumed that higher-mode solutions exist

- **In the present work based on the Fission Matrix:**

- We provide evidence that higher modes exist, are real, have discrete real eigenvalues, and are very nearly self-adjoint (for reactor-like problems)

- Approach is similar to Birkhoff's original proof for fundamental mode

- This has never been done before using continuous-energy Monte Carlo

# Higher Eigenmode Analysis with the Fission Matrix

---

- **Run Monte Carlo, get fission matrix, then solve for eigenvalues & eigenfunctions:**
  - Matlab, if full-storage F matrix can fit in memory
  - Power iteration with deflation, preserves sparse format
  - Implicitly Restarted Arnoldi Method (IRAM), preserves sparse format

$$\vec{S}_n = \frac{1}{K_n} \cdot \bar{F} \cdot \vec{S}_n \quad k_0 > |k_1| > |k_2| \dots > |k_N|$$

$$\vec{S}_n^\dagger = \frac{1}{K_n} \cdot \bar{F}^T \cdot \vec{S}_n^\dagger \quad n = 0, 1, \dots, N$$

$$(k_p - k_q) \cdot (\vec{S}_p \cdot \vec{S}_q^\dagger) = 0$$

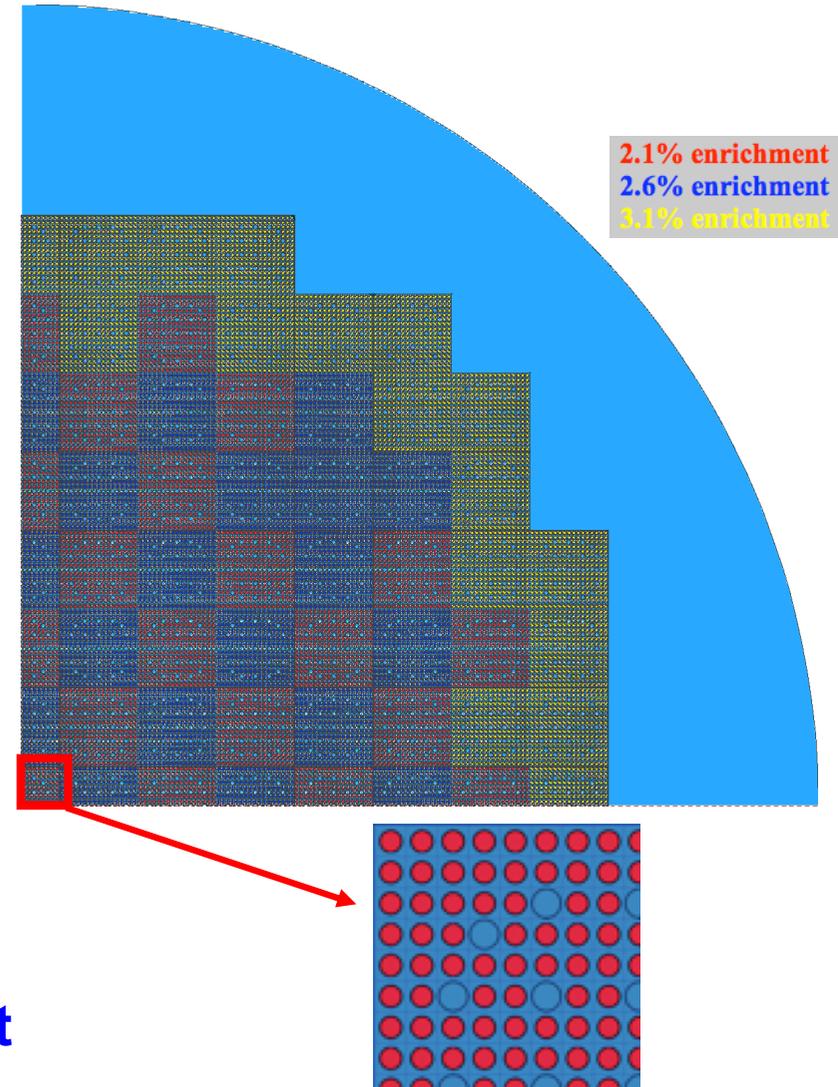
- F is nonsymmetric
- $S_n$  is a right eigenvector of F,  $S_n^\dagger$  is a left eigenvector of F
- $S_n$  and  $S_m^\dagger$  are biorthogonal

# Whole-core 2D PWR Model

## 2D PWR

(Nakagawa & Mori model)

- **48 1/4 fuel assemblies:**
  - 12,738 fuel pins with cladding
  - 1206 1/4 water tubes for control rods or detectors
- **Each assembly:**
  - Explicit fuel pins & rod channels
  - 17x17 lattice
  - Enrichments: 2.1%, 2.6%, 3.1%
- **Dominance ratio ~ .98**
- **Calculations used whole-core model, symmetric quarter-core shown at right**
- **ENDF/B-VII data, continuous-energy**



# Fission Matrix Analysis of PWR Model

---

- **Next 2 slides:**

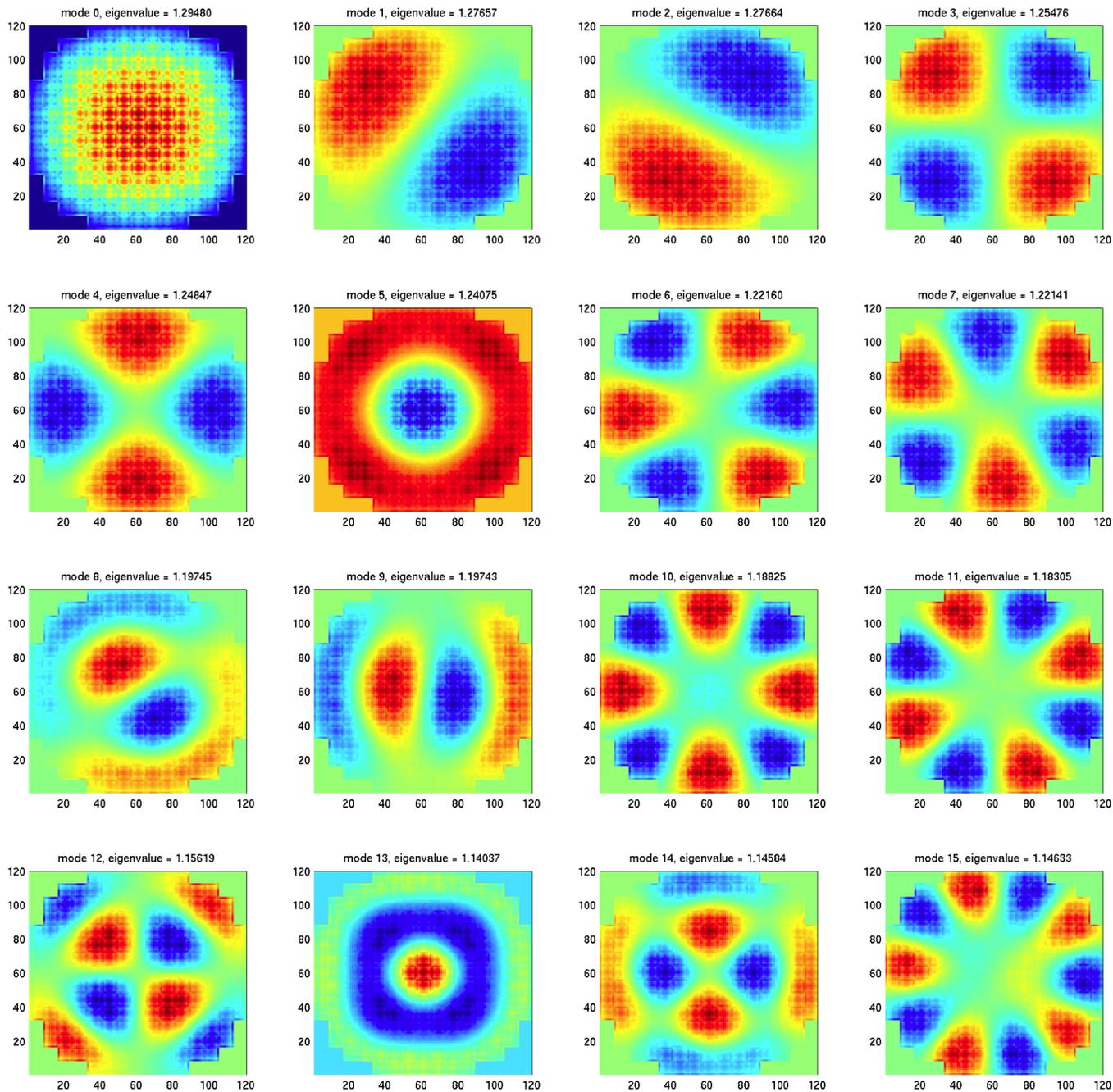
- **Spatial mesh for fission matrix:**

- 8 x 8 x 1 mesh per assembly
- 120 x 120 x 1 overall mesh
- **14,400 spatial regions**

- **Eigenvalues & eigenfunctions from Matlab:**

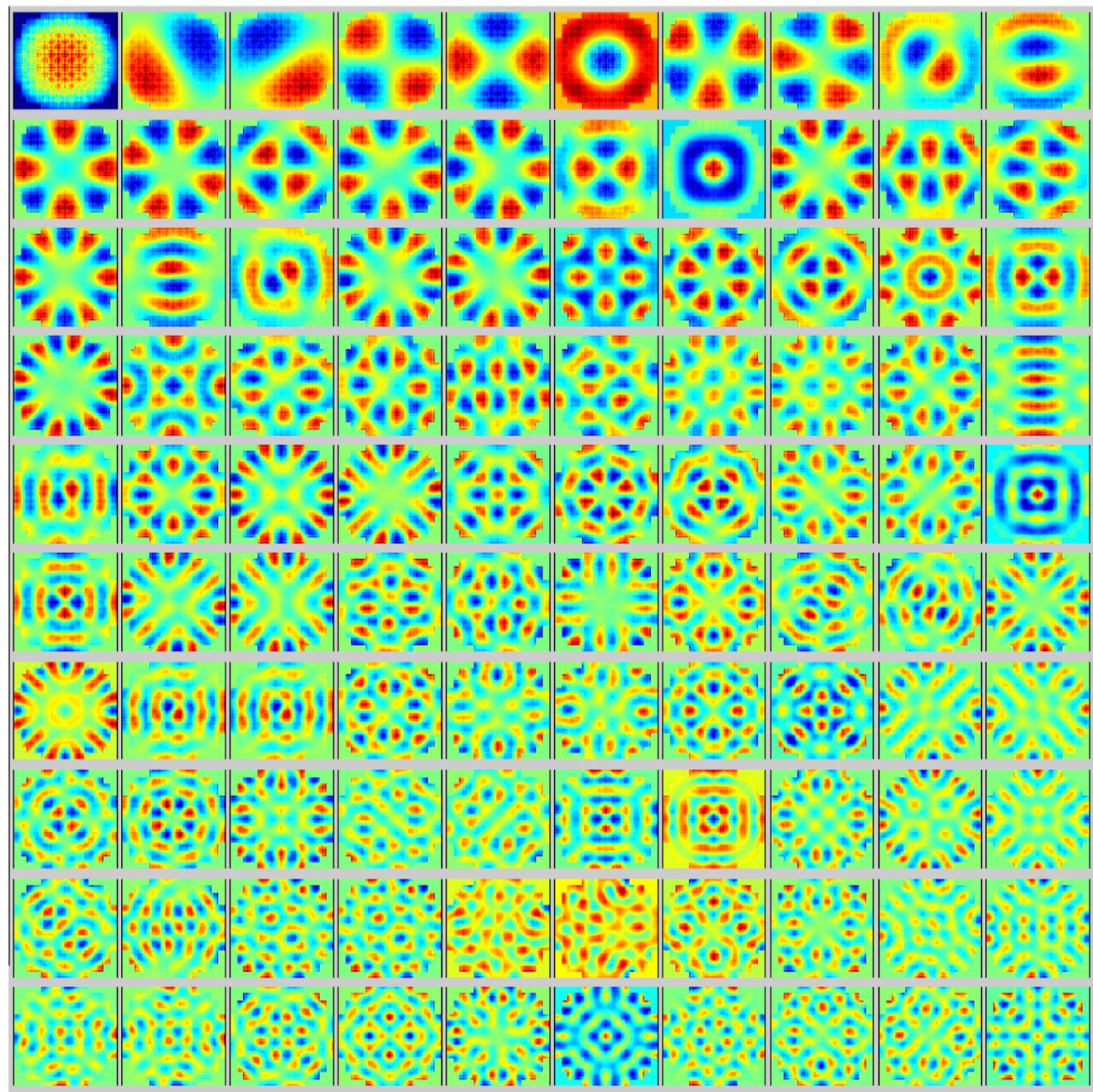
- For this **specific fission matrix size** of 14,400 x 14,400
- Fission matrix has 207 M elements = 1.6 GB
- Use Matlab to get all 14,400 eigenvalues & eigenvectors
  - Expensive, time-consuming – requires nonsymmetric eigensolver

# PWR – Eigenmodes for 120x120x1 Spatial Mesh



$n$	$K_n$
0	1.29480
1	1.27664
2	1.27657
3	1.25476
4	1.24847
5	1.24075
6	1.22160
7	1.22141
8	1.19745
9	1.19743
10	1.18825
11	1.18305
12	1.15619
13	1.14633
14	1.14617
15	1.14584

# PWR – First 100 Eigenmodes, with More Neutrons

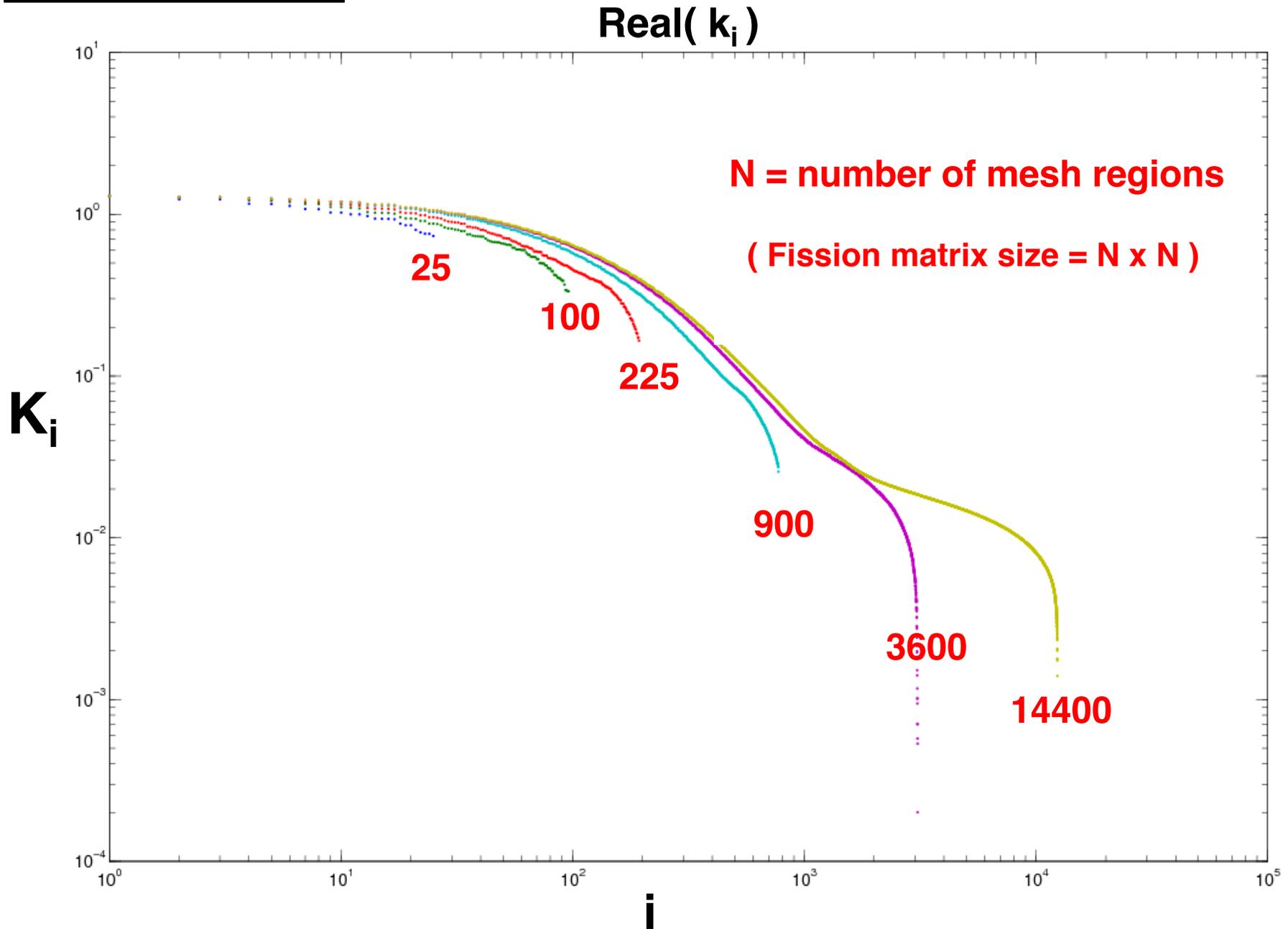


# Fission Matrix Analysis of PWR Model

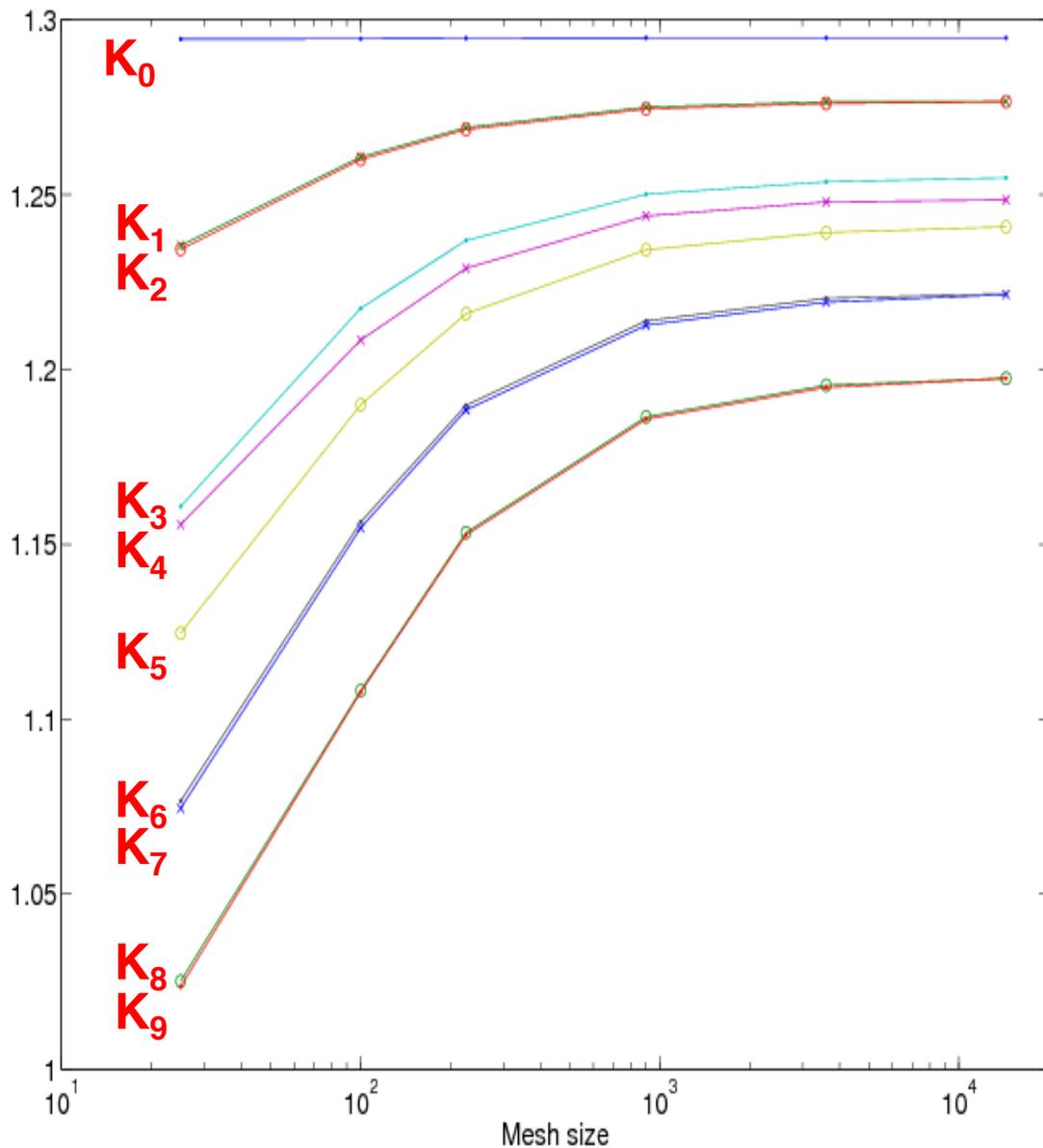
---

- **Following 2 slides:**
  - Vary the spatial discretization
  - Find eigenvalue spectrum for each discretization
  - **Examine eigenvalue spectrum vs number of spatial regions**
    - N regions  $\Rightarrow$  N eigenvalues
    - For small N, fewer eigenvalues to represent problem, inaccurate
  - **As N increases, spectrum extends & converges smoothly**
    - No anomalies, no oscillations
    - Provides measure of adequate mesh refinement for fission matrix accuracy

# Eigenvalue Spectra with Varying Meshes



# Spectrum Convergence from Mesh Refinement

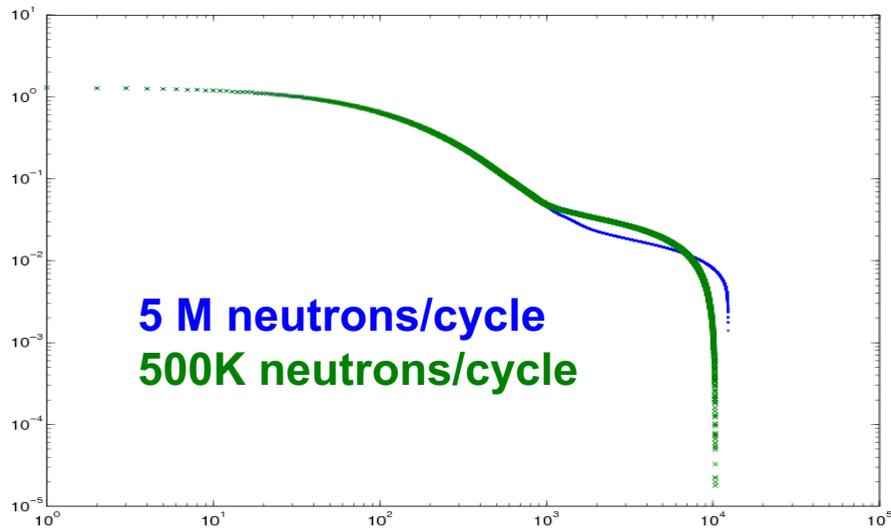


# Mesh Regions		$K_0$
5x5	= 25	1.29444
10x10	= 100	1.29453
15x15	= 225	1.29469
30x30	= 900	1.29477
60x60	= 3600	1.29479
120x120	= 14400	1.29480

**For fine-enough spatial mesh, eigenvalue spectrum converges**

# Are the Eigenvalues Real or Complex ?

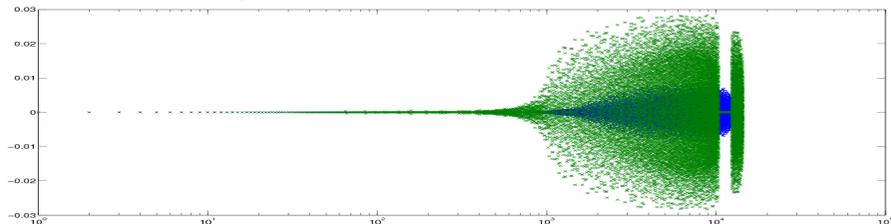
Real(  $k_i$  ):



The appearance of complex eigenvalues appears to be strictly an artifact of Monte Carlo statistical noise

When more neutrons/cycle are used to decrease statistical noise, complex components diminish or vanish

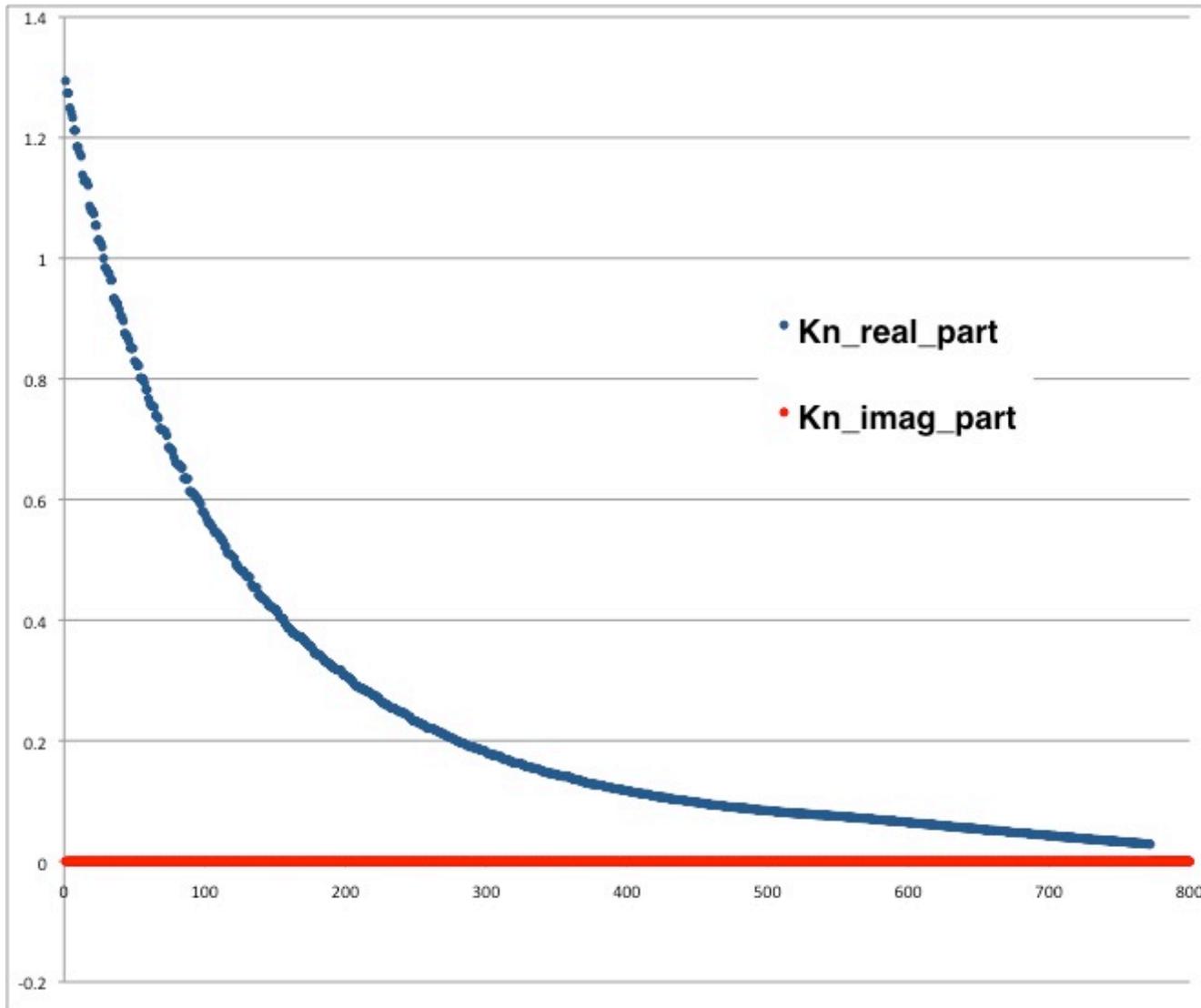
Imag(  $k_i$  ):



The first few 100s or 1000s of discrete eigenvalues are real, and presumably all would be with sufficiently large neutrons/cycle

120 by 120 Spectrum, Varying Neutrons/cycle

# PWR2D - Eigenvalues



**Fission Matrix**  
**30 x 30 mesh**

**772 Eigenvalues**

**2500 M neutrons**

**$K_n$  – real part**

**$K_n$  – imaginary part**

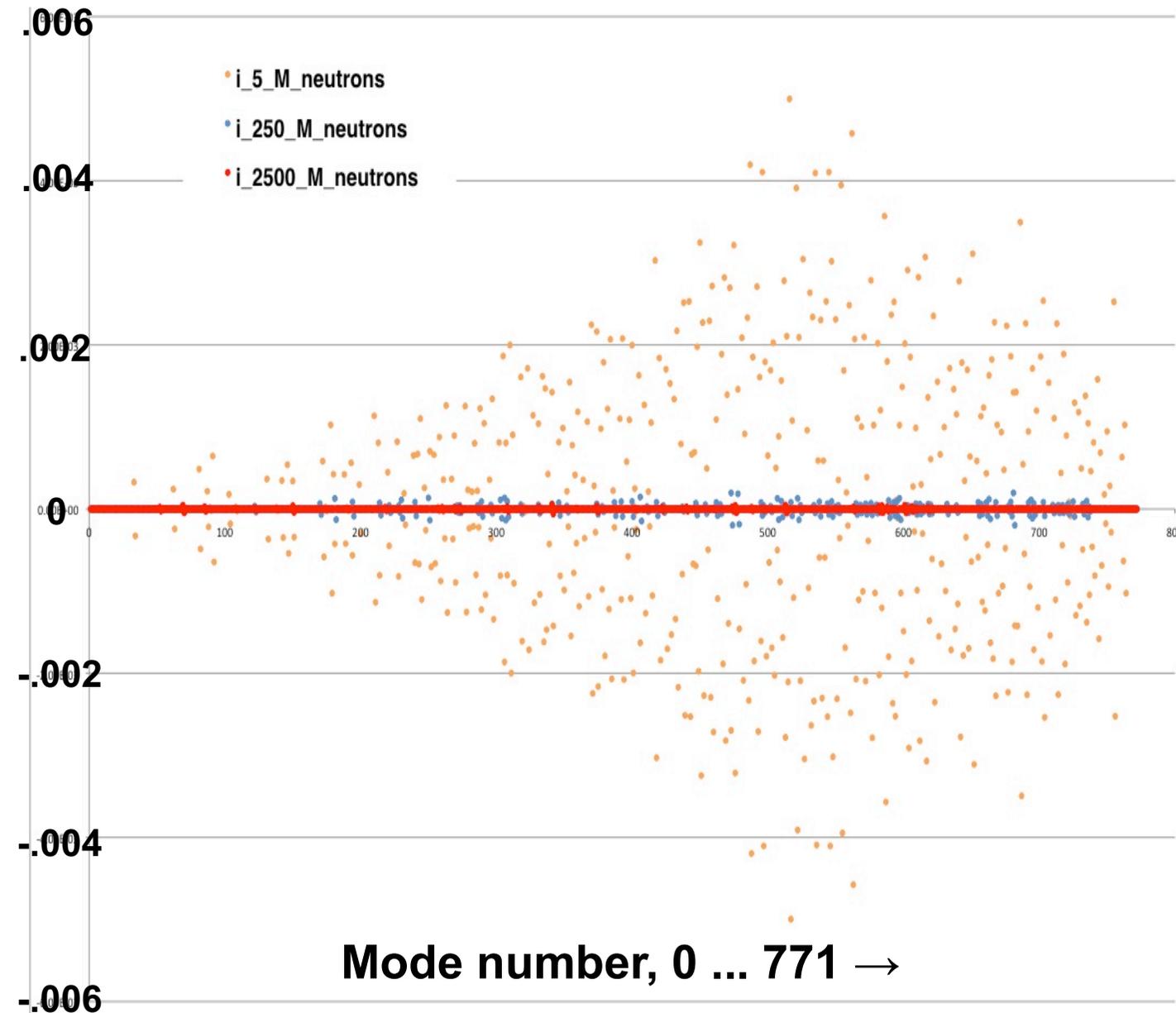
# PWR2D – Imaginary Part of Eigenvalues

Fission Matrix  
30 x 30 mesh

772 Eigenvalues

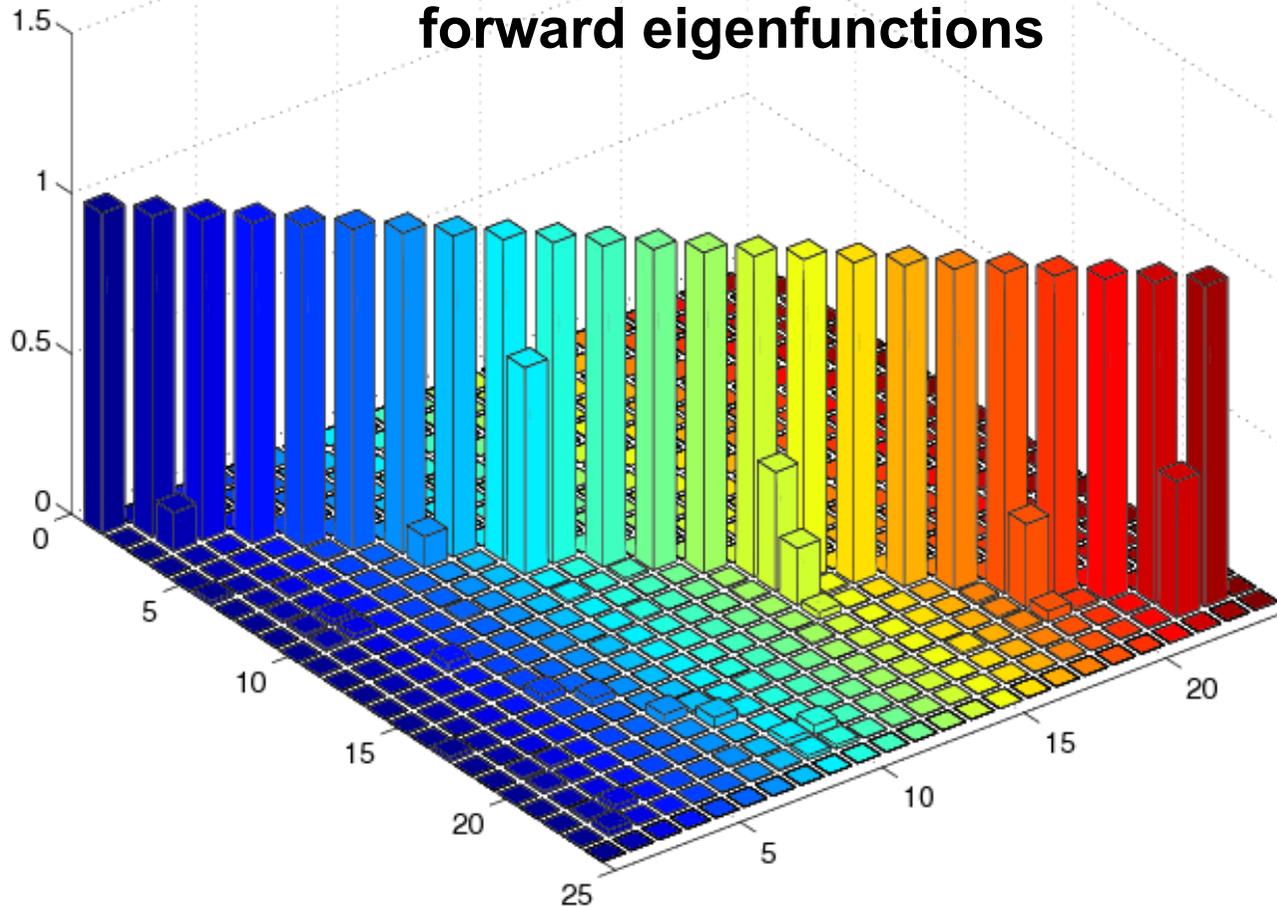
5 M neutrons  
250 M neutrons  
2500 M neutrons

$K_n$  – imaginary part



# PWR – Inner Products of Forward Eigenmodes

Inner products of  
forward eigenfunctions



$$\int d\vec{r} \psi_n(\vec{r})\psi_m(\vec{r})$$

$= \delta_{nm}$  if fission kernel  
is self adjoint/symmetric

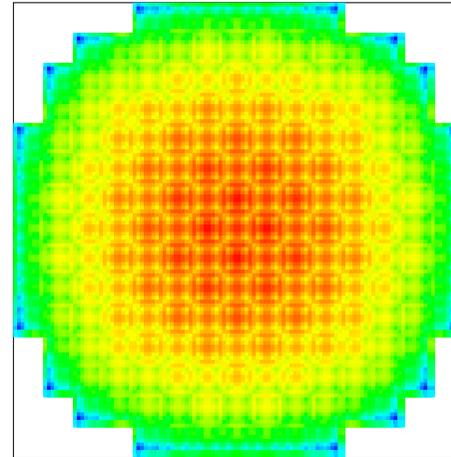
Strictly, eigenfunctions of the transport equation are bi-orthogonal.  
As shown above, **forward eigenfunctions are very nearly orthogonal.**

# PWR2D – Forward & Adjoint Source Eigenmodes

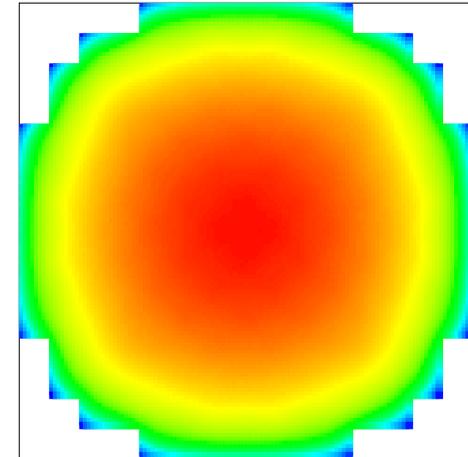
- **Fundamental eigenmode**

- Forward shows spatial detail, much like thermal flux
- Adjoint is smoother, much like fast flux

**Forward**

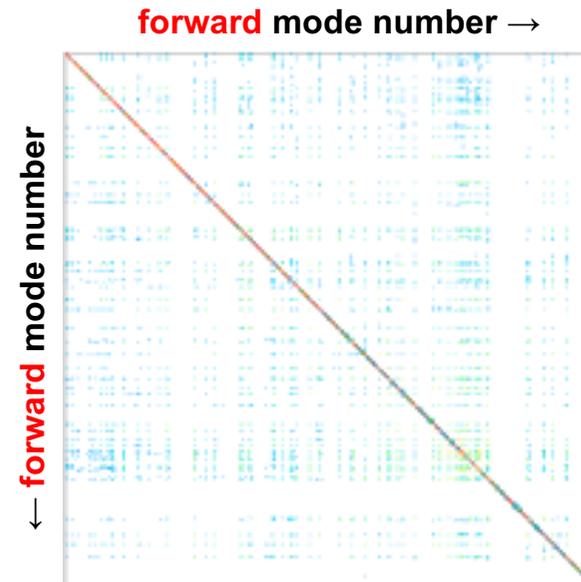
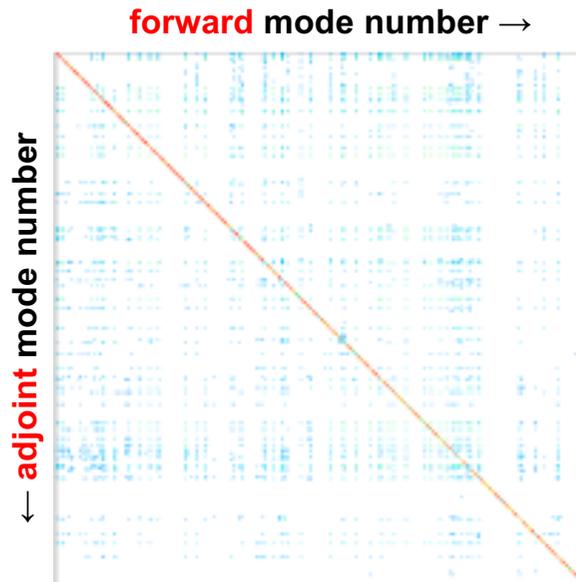


**Adjoint**



- **Inner products of modes:**

$$\mathbf{S}_n^t \cdot \mathbf{S}_m \quad \text{and} \quad \mathbf{S}_n \cdot \mathbf{S}_m$$

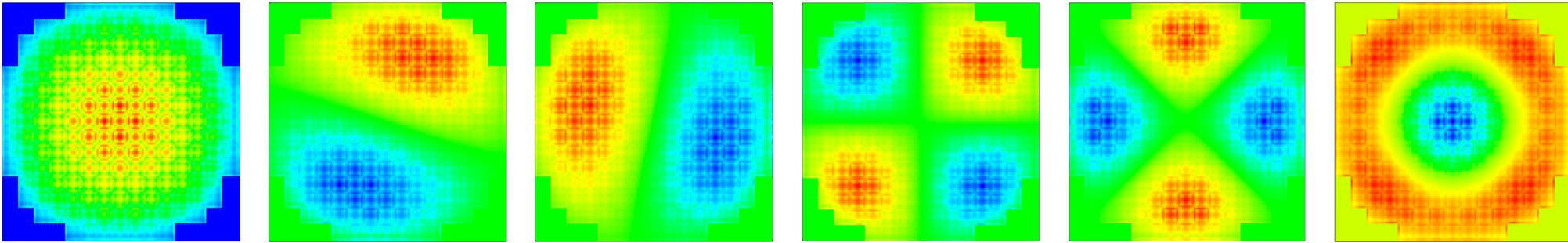


# Forward & Adjoint Source Eigenmodes

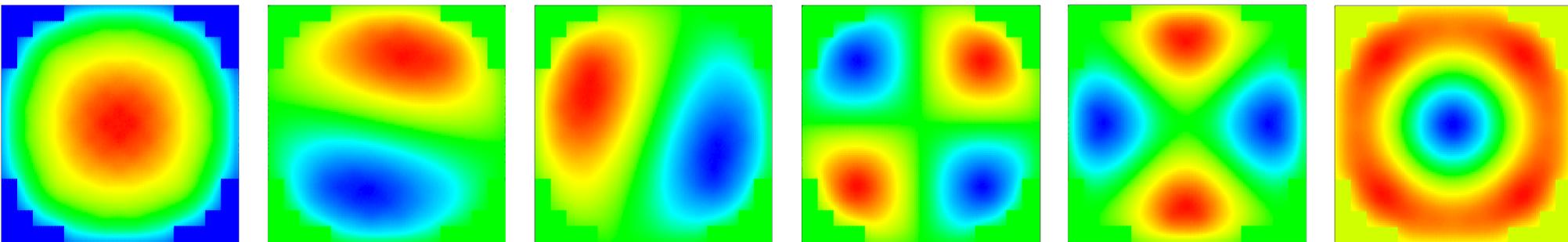
---

2D PWR problem – 2,500 M neutrons,  
tally mesh 120x120x1, matrix NxN N=14,400

## Forward source modes



## Adjoint source modes



0

1

2

3

4

5

# Calculation of Forward Flux Modes

---

- **Forward flux modes**

- Calculated by running **fixed source calculations** using forward fission source eigenfunctions

$$\Psi_n(\vec{r}, E, \hat{\Omega}) = \frac{1}{K_n} \mathbf{M}^{-1} \cdot \frac{\chi(\vec{r}, E)}{4\pi} S_n(\vec{r}) \quad n = 0, \dots, N$$

- **Source for mode  $n$  is sampled in an analog manner**

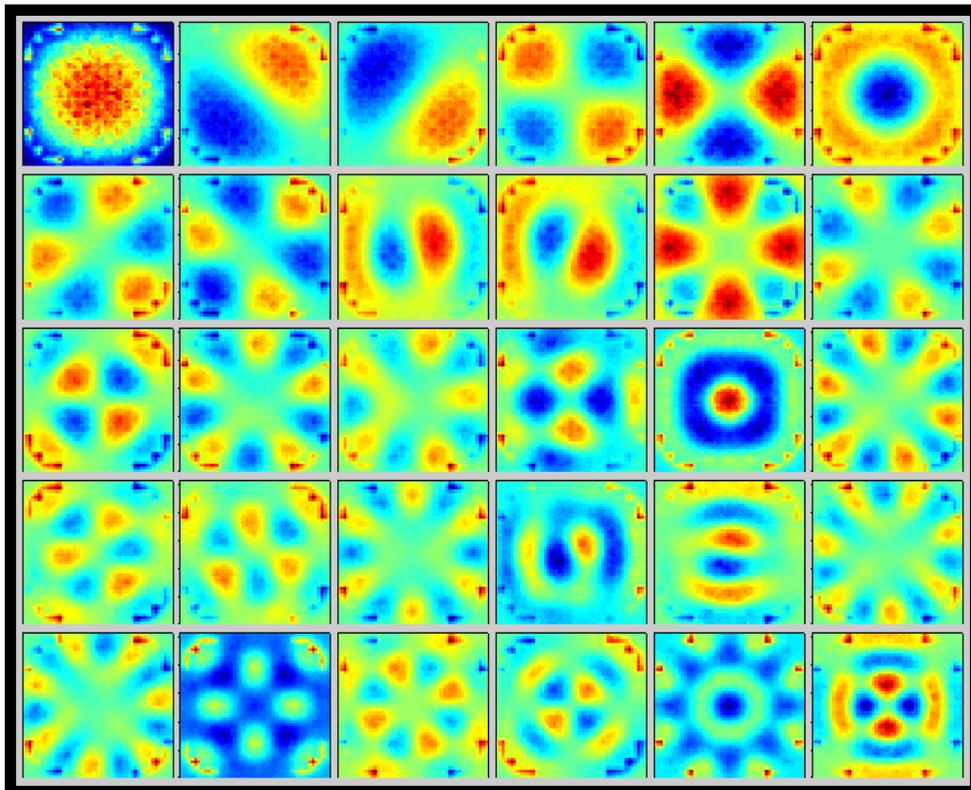
- Point within mesh cell is resampled until within fissionable material
- Flag is added for sign of particle weight
- Fission is treated as absorption (NONU card)

- **Track-length flux mesh tally module FMESH used**

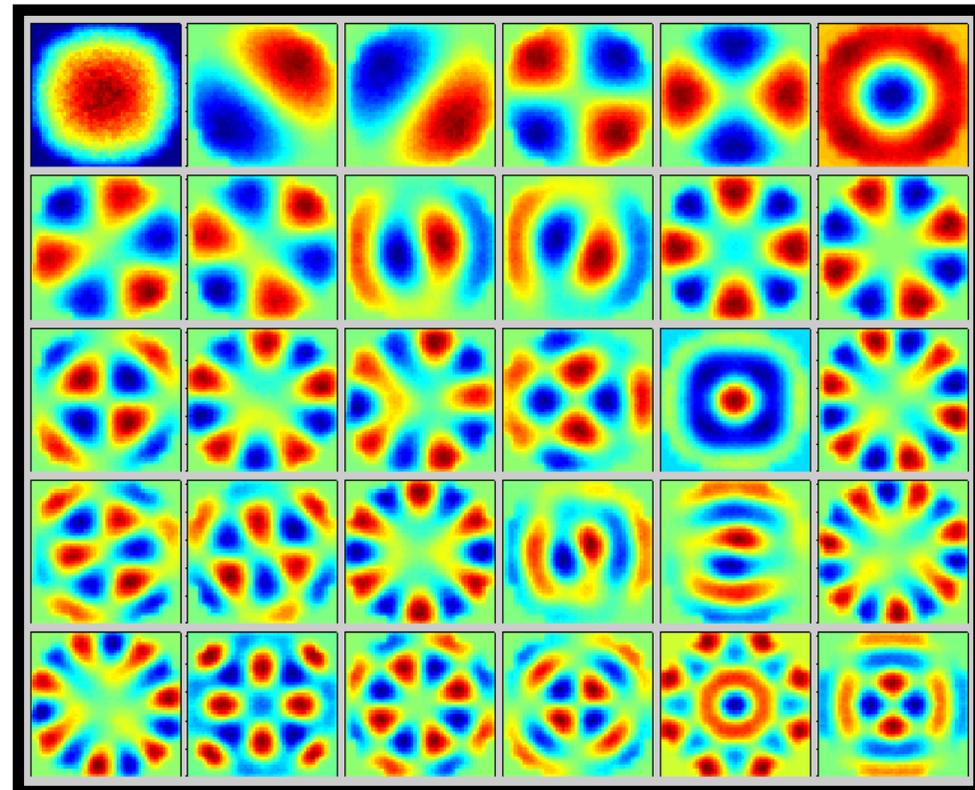
# Forward Flux Modes for 2D PWR

- Source modes from fission matrix
  - 500 cycles, 500k batch size
  - 50x50x1 mesh, 2500x2500 fission matrix
- Fixed source calculations
  - 500k histories per mode, minutes

Thermal flux modes (0 - 0.625 eV)

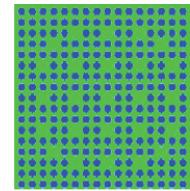
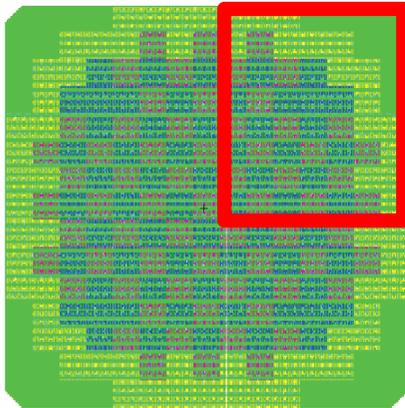


Fast flux modes (0.625 eV - 20 MeV)

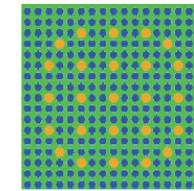


# PWR – with Perturbations

- Insert SS304 Control Rods in each assembly in quadrant of core



Original

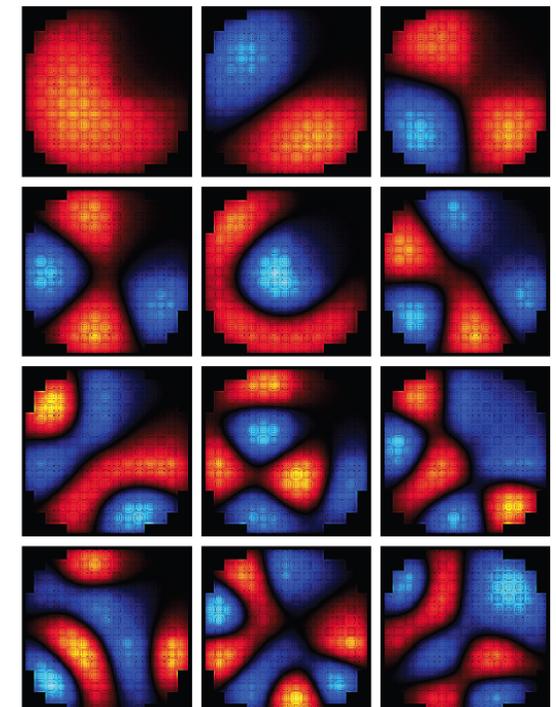
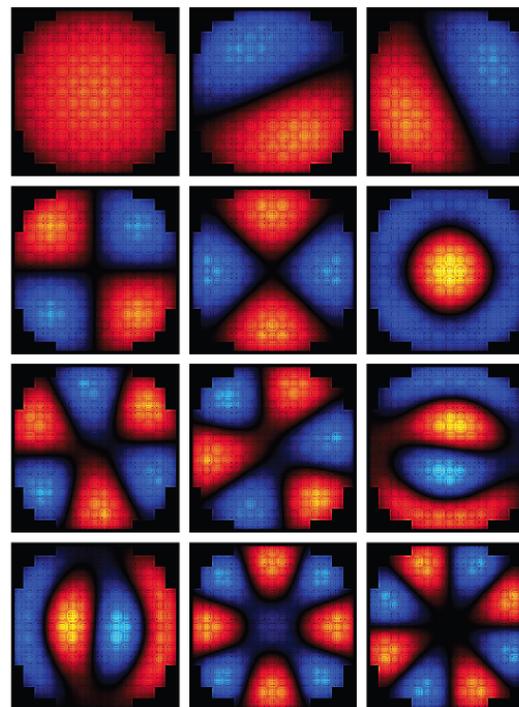


Perturbed

Original

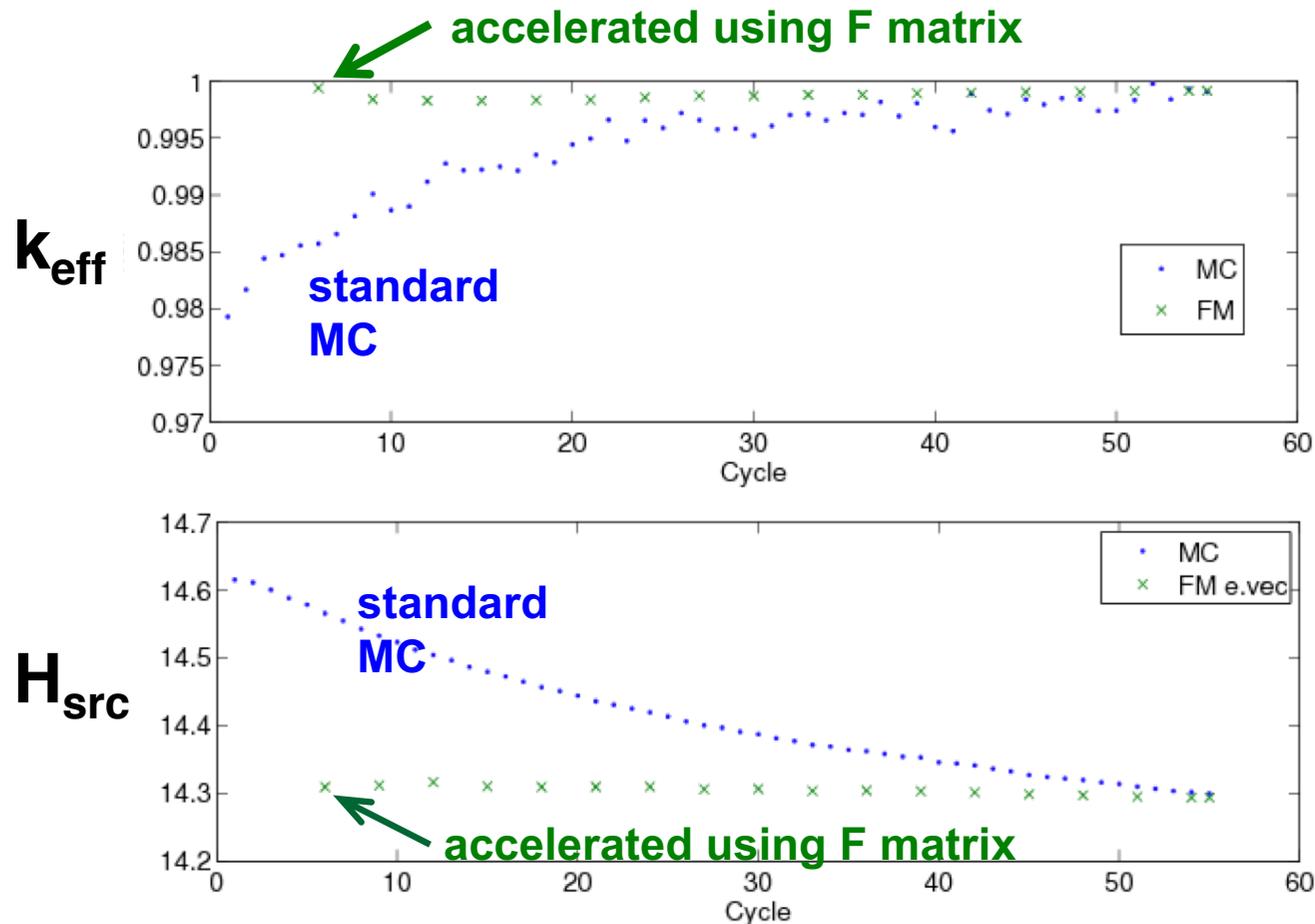
Perturbed

Fission Source  
Eigenmodes



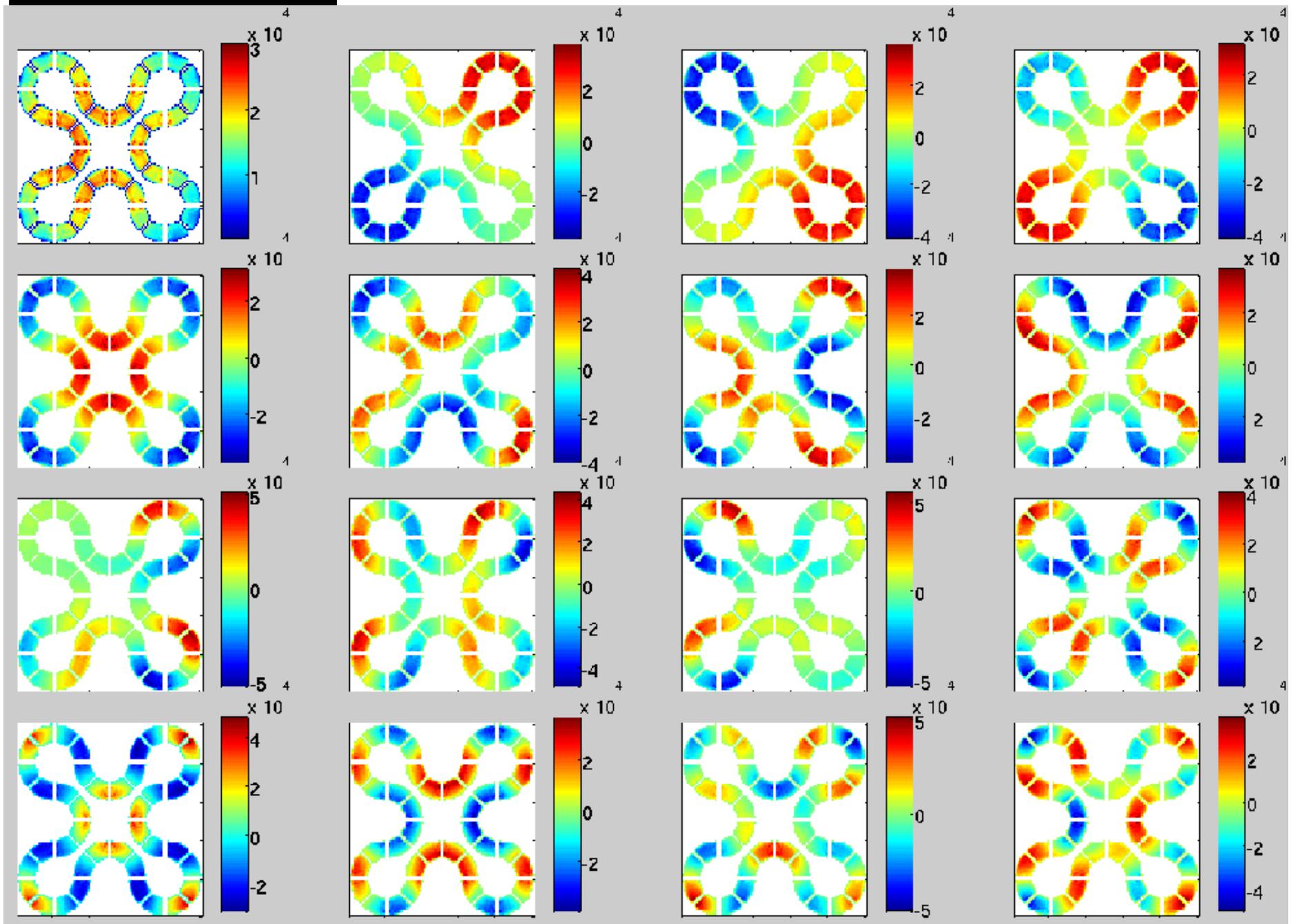
## PWR - Convergence Acceleration Using Fission Matrix

- Fission matrix can be used to **accelerate convergence** of the MCNP neutron source distribution during inactive cycles
- Requires only fundamental forward mode
- Very impressive convergence improvement





# ATR - Eigenmodes (100x100 spatial mesh)



# Conclusions & Future Work

# Conclusions

---

- **Derived theory underlying fission matrix method**
  - Rigorous Green's function approach, no approximations
  - Specific conditions on spatial resolution required for fission matrix accuracy
  - If spatial resolution fine enough, adjoint fission matrix identical to transpose of forward fission matrix
- **Applied to realistic continuous-energy MC analysis of typical reactor models. Numerical evidence that:**
  - Infinite set of discrete, real-valued eigenvalues & eigenfunctions exist for the integral fission neutron source & adjoint
  - As spatial resolution is refined, eigenvalue spectrum converges smoothly
  - While forward & adjoint are biorthogonal, forward modes are very nearly self-adjoint (for reactor-like problems)

# Conclusions

---

- **Fission matrix capability has been added to MCNP (R&D for now)**
- **Tested on variety of real problems (3D, continuous-energy)**
- **Can obtain fundamental & higher eigenmodes**
  - **Empirical evidence for:** **existence of higher modes, real, discrete eigenvalues, very nearly orthogonal eigenmodes (for reactor-like problems)**
  - **Higher eigenmodes are important for**
    - BWR void stability, Xenon oscillations, control rod worth, accident behavior,**
    - higher-order perturbation theory, quasi-static transient analysis, correlation effects on statistics, etc., etc., etc.**
- **Can provide very effective acceleration of source convergence**

# Future Work

---

- **Capabilities discussed in this talk are NOT in MCNP6.2 – targeted for release in later update**
- **Use fission matrix to accelerate source convergence**
  - Already demonstrated; very effective; needs work to automate
- **Use fission matrix for automatic, on-the-fly determination of source convergence**
  - Automate the determination of “inactive cycles”
- **Use fission matrix to assess problem coverage**
  - Need more neutrons/cycle to get adequate tallies?
- **Higher modes can be used to reduce/eliminate cycle-to-cycle correlation bias in statistics**
  - Replicas & ensemble statistics may be better, for exascale computers
- **Apply higher-mode analysis to reactor physics problems**
  - Xenon & void stability, slow transients, etc.

# References

---

- Carney, Brown, Kiedrowski, Martin, “Fission Matrix Capability for MCNP Monte Carlo”, TANS 107, San Diego, 2012
- Brown, Carney, Kiedrowski, Martin, “Fission Matrix Capability for MCNP, Part I - Theory”, M&C-2013, 2013
- Carney, Brown, Kiedrowski, Martin, “Fission Matrix Capability for MCNP, Part II - Applications”, M&C-2013, 2013
- Brown, Carney, Kiedrowski, Martin, “Fission Matrix Capability for MCNP Monte Carlo”, SNA+MC-2013
- Carney, Brown, Kiedrowski, Martin, “Theory & Application of the Fission Matrix Method for Continuous-Energy Monte Carlo”, Annals Nuc. En. 73, 423-431 (2014)

**Questions ?**



**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Continuously Varying Material Properties & Tallies for Monte Carlo Calculations**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



## Abstract

---

# Continuously Varying Material Properties and Tallies For Monte Carlo Calculations

Forrest B. Brown (LANL), David P. Griesheimer, & William R. Martin (U. Mich)

Monte Carlo methods for particle transport are highly regarded for their continuous treatment of particle energy and angular dependence, and for their very general geometric representations. However, all of the production Monte Carlo codes available today make use of a zero-th order approximation for representing material densities and cell flux tallies, i.e., constant over a geometric cell. Recent work has shown that both of these limitations can be overcome, so that continuously varying spatial representations can be extended to material properties and tallies.

In the present work, we provide derivations of the new random sampling methods and mathematical procedures and then demonstrate the feasibility of these new methods in a 1-D Monte Carlo code. We have used this code first to verify that the continuous representation was implemented correctly, and then to investigate a number of deep penetration problems and eigenvalue problems to examine the benefits of a continuous representation.

The theory and numerical results described herein demonstrate conclusively that it is now feasible to implement Monte Carlo codes with continuously varying particle energy and angular dependence, continuously varying material properties, and continuously varying tallies. The continuous representation can greatly reduce modeling difficulties, can significantly reduce the number of cells required for accurate results, and for complex problems may even reduce the computation time.

# Outline

---

- **Introduction**
- **Varying Material Properties**
  - Stepwise Approximation
  - Woodcock Tracking
  - Direct Numerical Sampling
  - Piecewise Legendre Expansion
- **Continuous Tallies**
  - Stepwise Approach
  - Piecewise Legendre Expansion
- **Numerical Examples**
  - Fixed Source Example
  - Criticality Example
  - Timing & Complexity
- **Conclusions**

# Introduction

---

- **Monte Carlo codes such as MCNP5 are continuous in**
  - Particle properties: position, direction, & energy
  - Collision physics: energy & angle
- **Monte Carlo codes permit very general 3D geometries & cross-section data representations**
- **BUT, Monte Carlo codes use zero-th order representations of tallies and material properties:**
  - Material properties are assumed constant within each cell
  - Tally bins provide average scores within each cell

**We should be able to do better than that !**

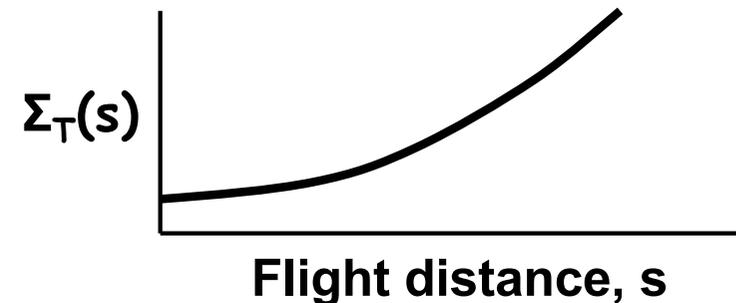
**This work demonstrates that we can.**

# Varying Material Properties

For many problems of interest,  $\Sigma_T$  varies within a cell

- **Charged particle transport**

- Continuous slowing down along the flight path due to interactions with electron field in material
- $\Sigma_T$  increases along the flight path



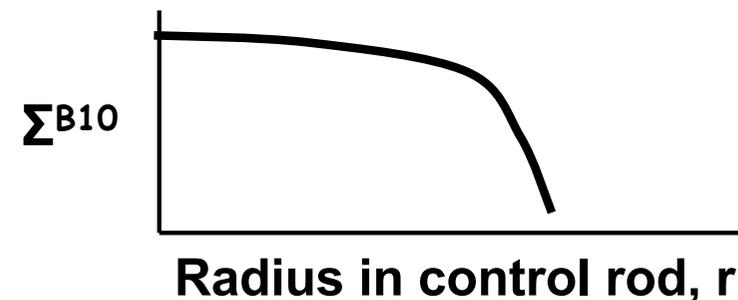
- **Atmospheric transport**

- Air density varies with altitude



- **Depleted reactor**

- Fuel & poison distribution varies due to burnup



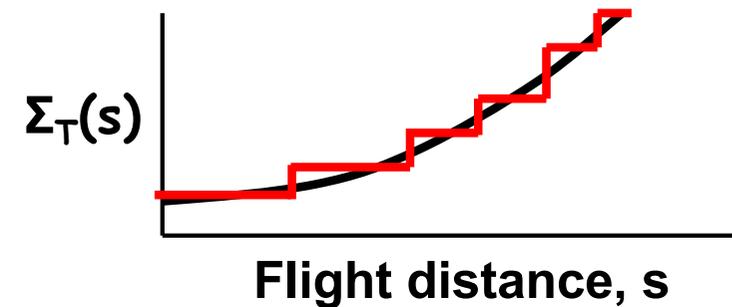
# Varying Material Properties

---

## Conventional techniques for handling varying material properties:

- **Stepwise approximation**

- Subdivide geometry
- Constant material properties within each step



- **Woodcock tracking**

- Also called delta tracking, fast tracking, pseudo-collision method, hole tracking, ...
- Involves biased sampling the flight distance using a larger  $\Sigma_T$ , followed by rejection sampling to assure a fair game

# Woodcock Tracking

- **Introduce  $\Sigma$  for a "delta" collision**

- Let  $\Sigma^* = \Sigma_T(s) + \Sigma_\delta(s) = \text{constant}$ ,

where  $\Sigma_\delta(s) \geq 0$

$\Sigma_\delta(s) =$  cross-section for "delta" collision -  
no change in  $E$ ,  $(u,v,w)$ , or wgt

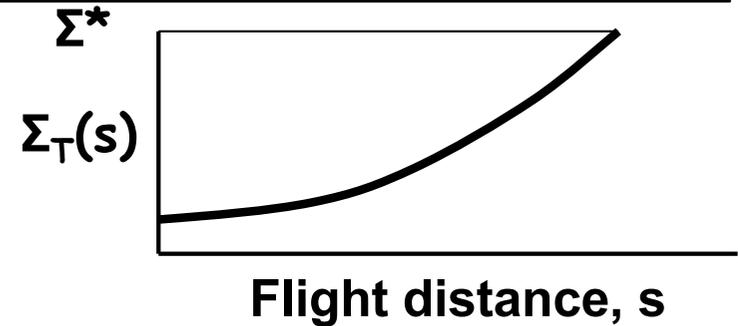
$$\Sigma^* \geq \Sigma_T(s)$$

- $\Sigma_T(s) / \Sigma^* =$  probability of a "real" collision

- $\Sigma_\delta(s) / \Sigma^* =$  probability of a "delta" collision

- **Basic idea: Sample flight distance using  $\Sigma^*$ ,**  
**move the particle to the collision point,**  
**then reject collision point if  $\xi > \Sigma_T(s) / \Sigma^*$**

- **Using  $\Sigma^*$  rather than  $\Sigma_T(s)$  gives an interaction probability per unit distance that is too large, hence a flight distance that is too short. Rejection scheme compensates for this.**



# Limitations on Conventional Techniques

---

- **Stepwise approximation**
  - How many steps? How small?
  - Accuracy vs number of steps
  - Need to perform convergence studies: results vs stepsize
  - Tedious to set up
  - More cells --> increases time for tracking & boundary crossing
- **Woodcock tracking**
  - Inefficient if  $\Sigma^* \gg \Sigma_T(s)$  for most values of  $s$
  - Can't use pathlength estimators for tallies

## Alternatives ?

Direct numerical sampling method

(Brown & Martin, Gatlinburg M&C Topical, 2003)

# Sampling the flight distance in varying media

---

- Random sampling of particle free-flight distance in media where the cross-sections are **constant** during the particle flights, solve for  $s$ :

$$\xi = \int_0^s \Sigma(x) \cdot \exp[-\Sigma x] dx$$

- Random sampling of particle free-flight distance in media where the cross-sections **vary** during the particle flights, solve for  $s$ :

$$\xi = \int_0^s \Sigma(x) \cdot \exp\left[-\int_{x_0}^x \Sigma(x') dx'\right] dx$$

# Sampling the flight distance in varying media

- **Optical depth along flight path**

$$\tau(s) = \int_x^{x+s} \Sigma_T(x') dx' \quad \Sigma_T(x) \text{ is finite, } \Sigma_T(x) \geq 0$$

Note that  $\frac{d\tau(s)}{ds} = \Sigma_T(x+s), \quad 0 \leq \frac{d\tau}{ds} \leq \infty$

- **To explicitly allow for the case of no collision,**

$P_{NC}$  = probability of no collision  $P_{NC} = e^{-\tau(\infty)}$

- **Probability density function (pdf) for the flight distance s:**

$$f(s) = P_{NC} \cdot \delta(s = \infty) + (1 - P_{NC}) \cdot \frac{1}{G} \frac{d\tau}{ds} e^{-\tau(s)}$$

Where  $G = \int_0^{\infty} \frac{d\tau(s)}{ds} e^{-\tau(s)} ds = 1 - e^{-\tau(\infty)} = 1 - P_{NC}$

## Sampling the flight distance in varying media

---

- **Random sampling of the Monte Carlo free-flight path requires solving the following equation for  $s$ , the flight path:**

$$\xi = \int_0^s f(x) dx$$

*or*

$$\xi = P_{NC} \cdot H(s, \infty) + (1 - P_{NC}) \cdot \frac{1}{G} \cdot \left(1 - e^{-\tau(s)}\right)$$

- **Common case:  $\Sigma_T$  independent of  $x$**

$$\tau(s) = \Sigma_T \cdot s, \quad \frac{d\tau}{ds} = \Sigma_T, \quad P_{NC} = 0, \quad G = 1, \quad f(s) = \Sigma_T \cdot e^{-\Sigma_T \cdot s}$$

**With solution:**

$$s = -\frac{\ln(1 - \xi)}{\Sigma_T}$$

# Sampling the flight distance in varying media

---

## Direct Numerical Sampling for the free-flight distance:

### Step [1]

If  $\xi < P_{NC}$ , Then: No collision, set  $s = \infty$ , exit  
 Otherwise: Do Steps 2 & 3

### Step [2]

Define  $\hat{\tau} = \tau(s)$

Sample  $\hat{\tau}$  by solving  $\xi = \frac{1}{G} \int_0^{\hat{\tau}} e^{-\tau} d\tau$ , with  $0 \leq \hat{\tau} \leq \tau(\infty)$

That is, sample from a truncated exponential PDF:

$$\hat{\tau} = - \frac{\ln(1 - \xi \cdot G)}{\Sigma_T}$$

### Step [3]

Solve for  $s$ :  $\hat{\tau} = \tau(s) = \int_0^s \Sigma_T(x + s') ds'$

Analytic solution if possible, otherwise use Newton iteration

# Sampling the flight distance in varying media

---

## Newton iteration to numerically solve for s:

$$s_0 = \hat{\tau} / \Sigma_T(x_0)$$

$$n = 0$$

*Iterate:*

$$n = n + 1$$

$$g = \hat{\tau} - \tau(s_{n-1})$$

$$g' = dg / ds = -\Sigma_T(x_0 + s_{n-1})$$

$$s_n = s_{n-1} - g / g'$$

$$\text{Stop if } |s_n - s_{n-1}| < \varepsilon$$

## Notes:

- Because  $g' < 0$ ,  $g(s)$  is monotone & there can be only one root
- For cases where  $\Sigma_T > 0$ , Newton iteration guaranteed to converge
- If  $\Sigma_T(x) = 0$  or very small,  $g'$  may be 0, leading to numerical difficulties
- Remedied by combining Newton iteration with bisection if  $g'$  near zero
- Typically only 1-5 iterations needed to converge  $s$  to within  $10^{-6}$

# Verification of Direct Numerical Sampling

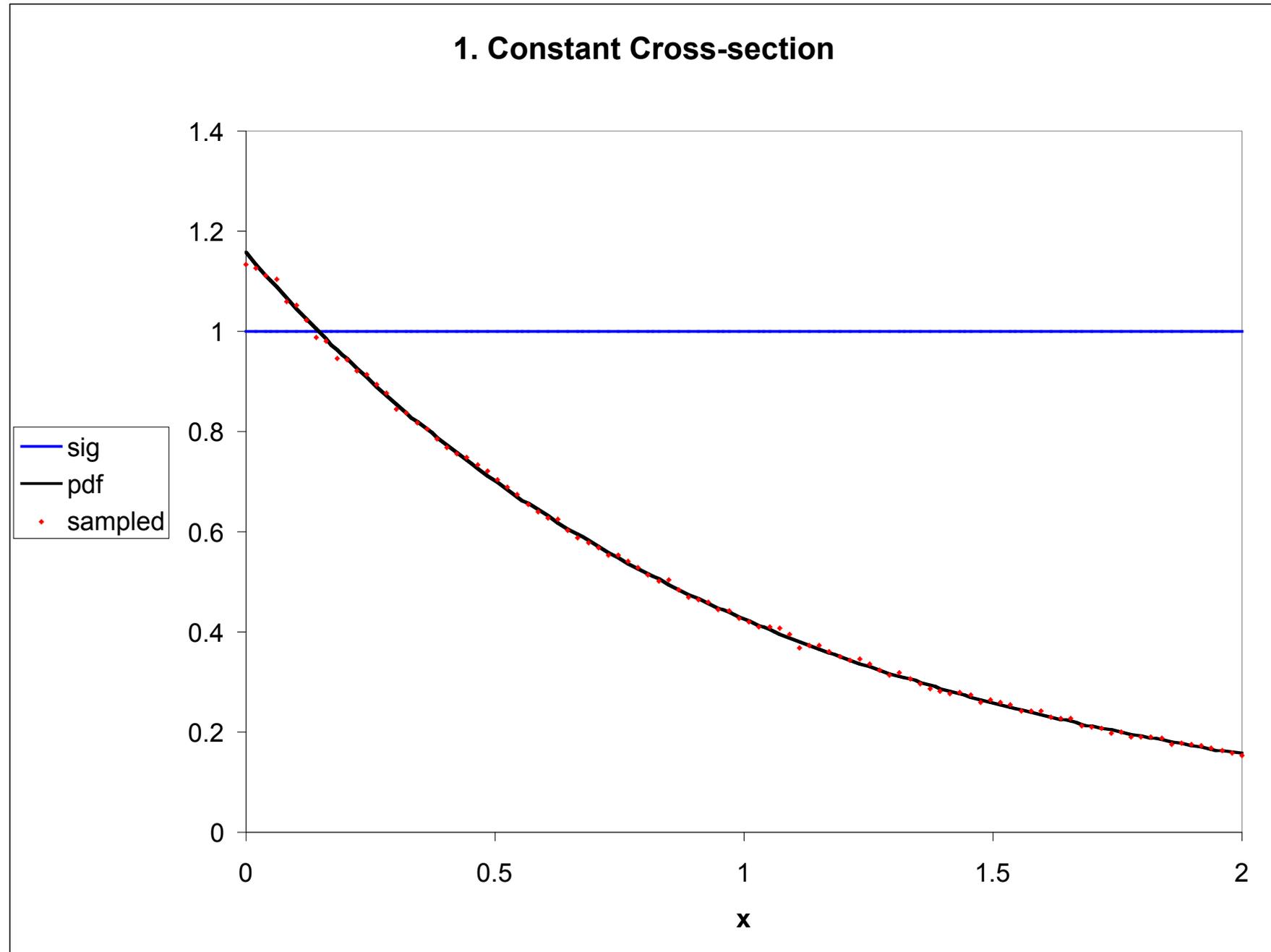
---

- Monte Carlo transport of particles through a 1-D slab of thickness 2 units.
- Consider only transmission through the slab, ignoring scattering.
- Table (1) shows 7 different forms of spatial variation in the cross-section which were used for the test problem.
- Figures (1) through (7) show the cross-section variation over the thickness of the slab (labelled "sig"), the pdf at position  $x=0$  for the cross-section variation in each test case (labelled "pdf"), and the results of using the direct numerical sampling procedure to perform 1,000,000 samples of the free-flight distance for each case (labelled "sampled").
- The sampled results were binned in 100 bins of width 0.02.
- In Figures (1)-(7), it can be seen that the distributions of sampled results for the free-flight distance agree completely with the exact pdf's in all cases, verifying that the sampling method is correct.

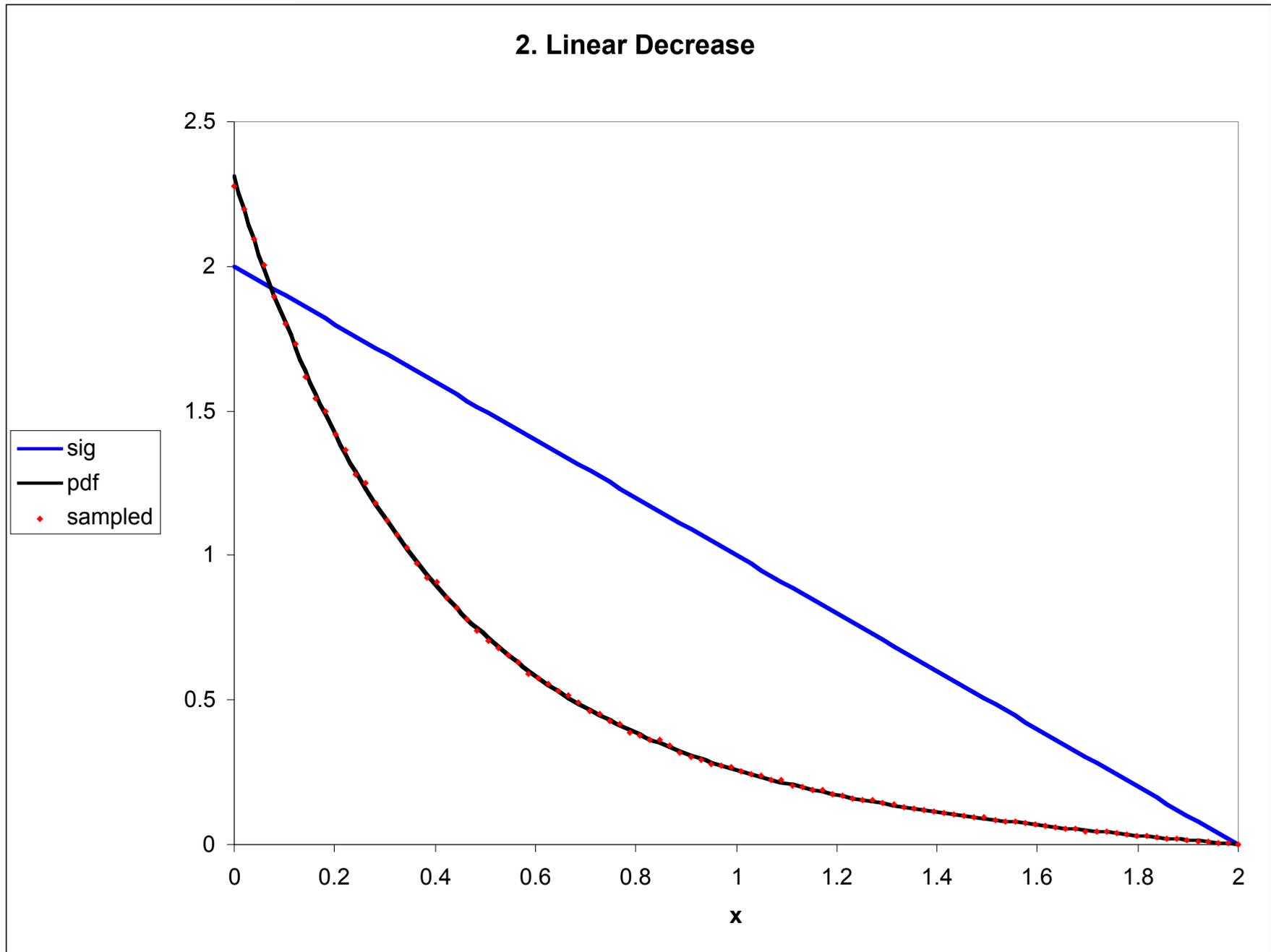
**Table 1. Cross-section Variation for Test Cases**

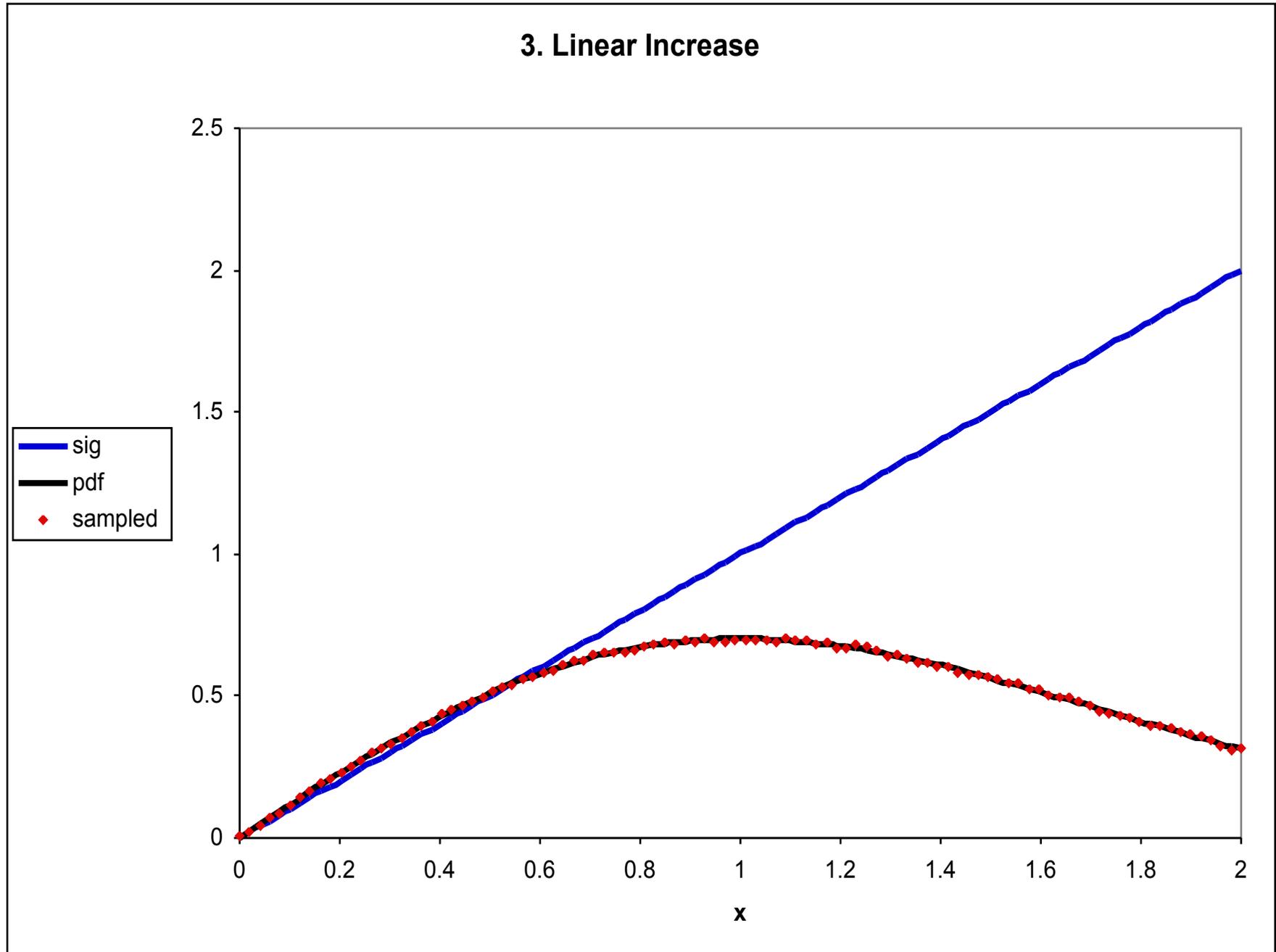
Case	Cross-section Variation	Numerical Representation for range $0 \leq x \leq 2$
1	Constant	$\Sigma(x) = 1$
2	Linearly Decreasing	$\Sigma(x) = 2 - x$
3	Linearly Increasing	$\Sigma(x) = x$
4	Exponentially Decreasing	$\Sigma(x) = \exp(-3x)$
5	Exponentially Increasing	$\Sigma(x) = 0.1 \cdot \exp(2x)$
6	Sharp Gaussian	$\Sigma(x) = \frac{2}{\sqrt{2\pi}} \exp \left[ - \left( \frac{x-1}{0.05} \right)^2 \right]$
7	Broad Gaussian	$\Sigma(x) = \frac{2}{\sqrt{2\pi}} \exp \left[ - \left( \frac{x-1}{1.0} \right)^2 \right]$

# Figure 1. Test Case - 1

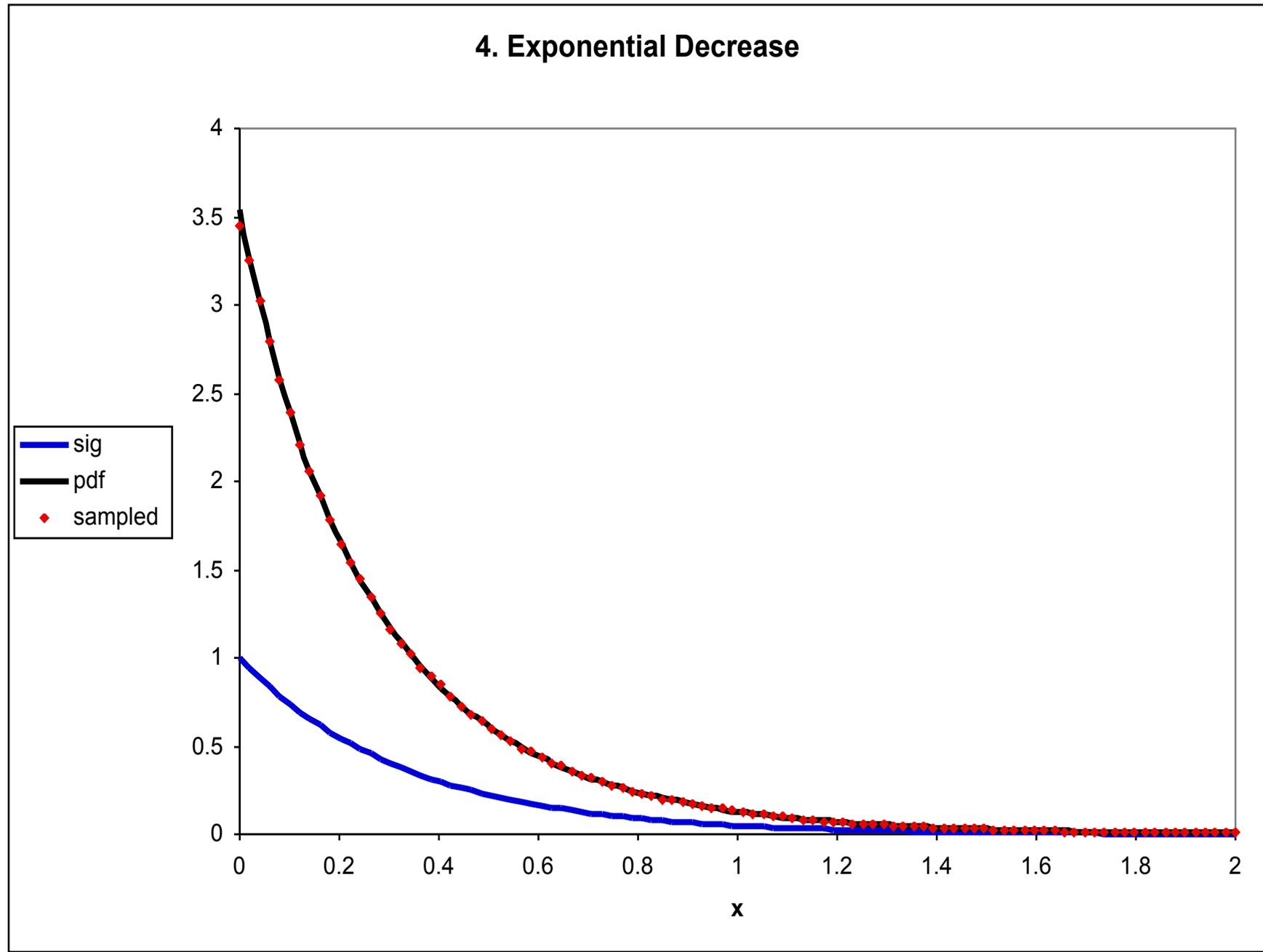


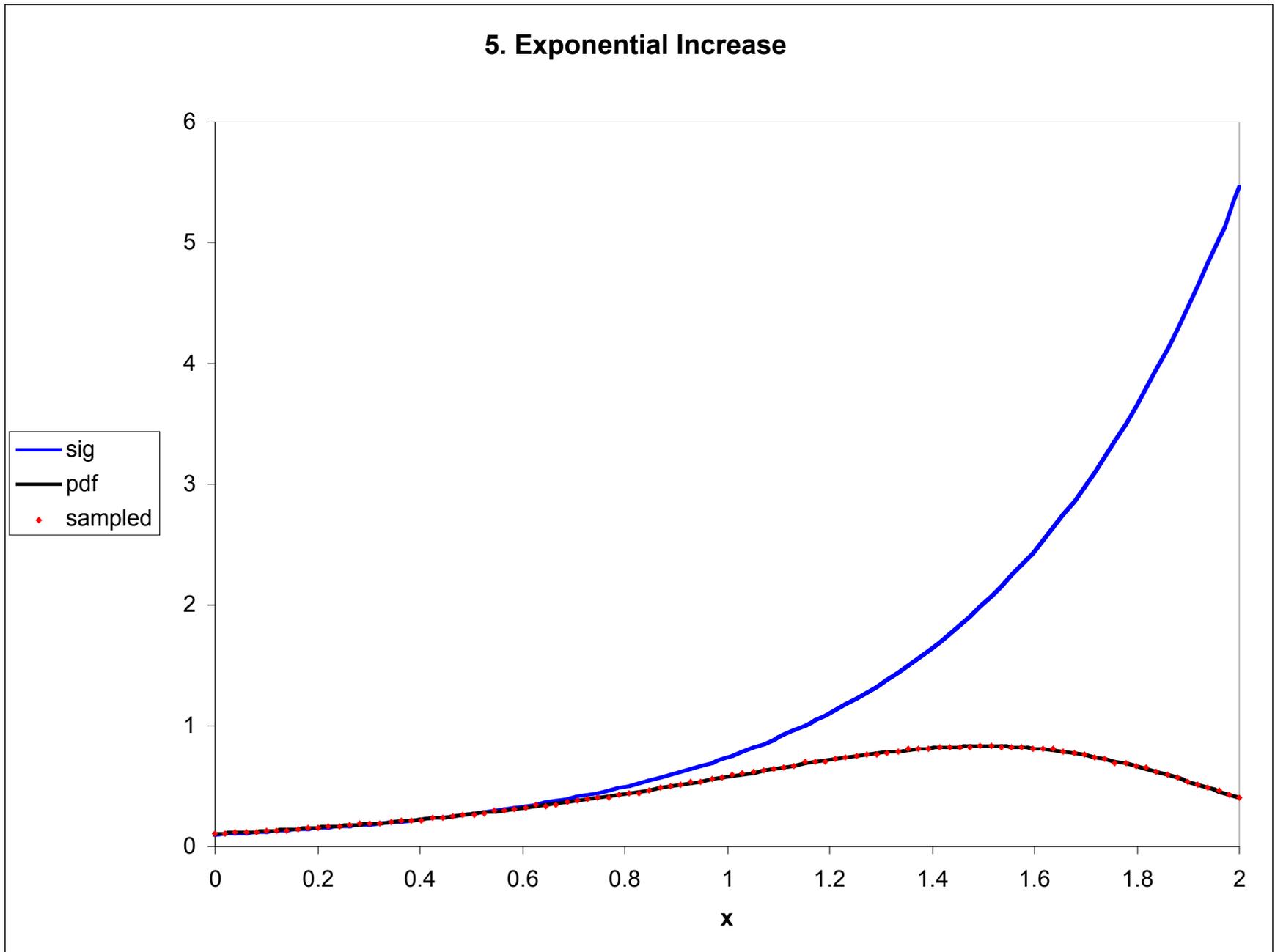
# Figure 2. Test Case - 2

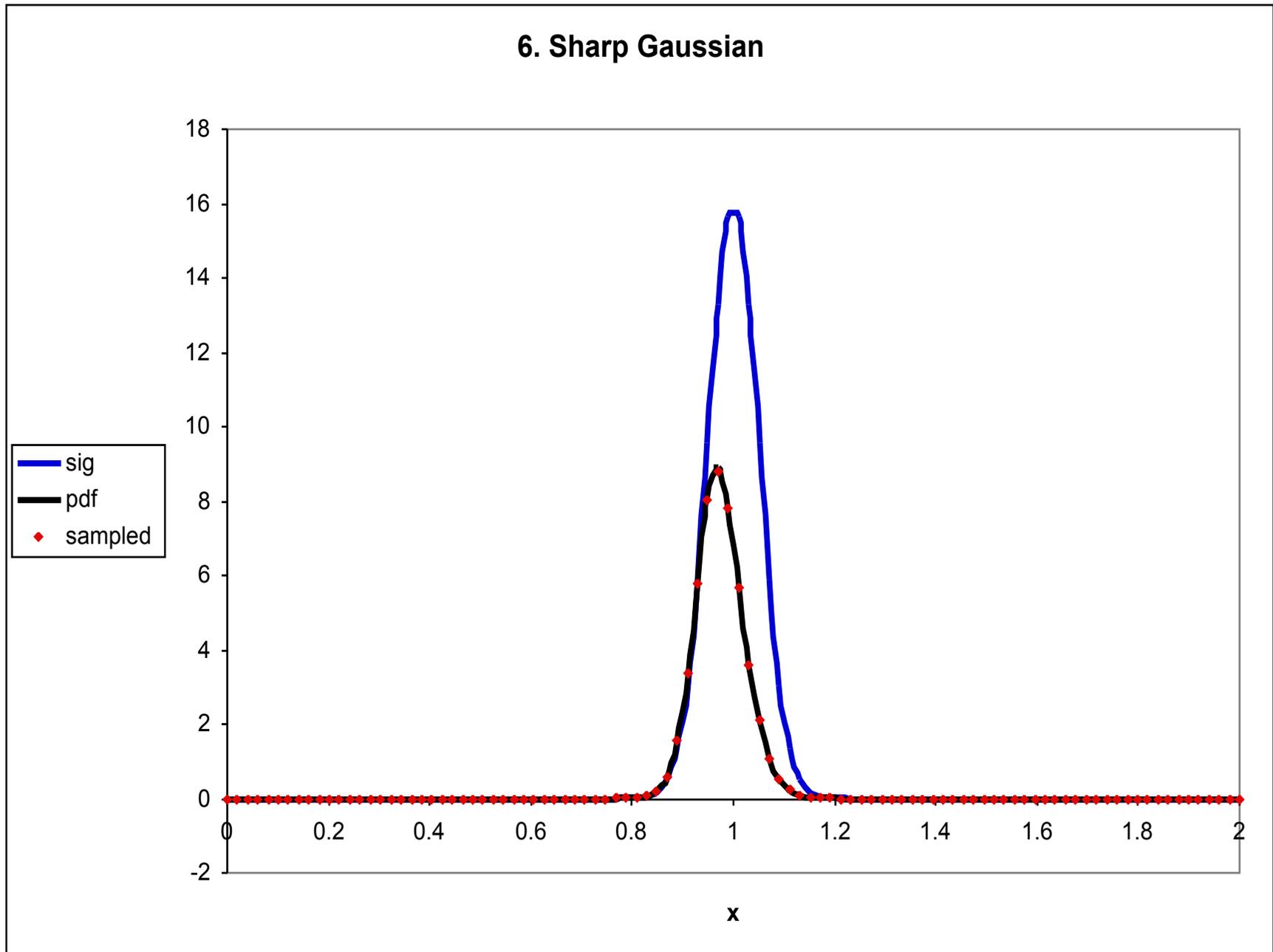


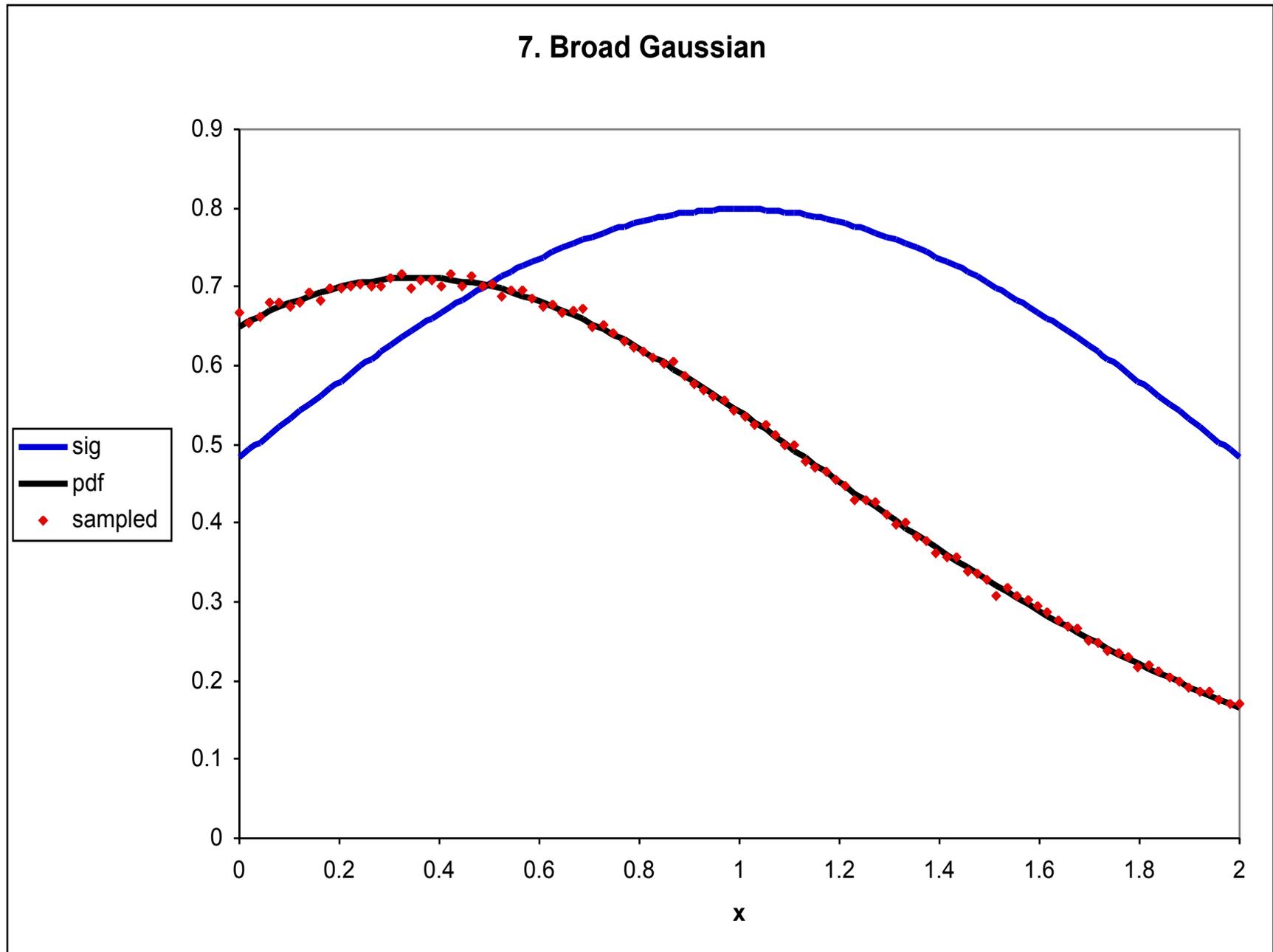
**Figure 3. Test Case - 3**

# Figure 4. Test Case - 4



**Figure 5. Test Case - 5**

**Figure 6. Test Case - 6**

**Figure 7. Test Case - 7**

## **Compare: Direct Numerical, Delta-tracking, & Substepping**

---

- **Multiple collisions were followed**
- **Transmission through the slab was computed for each test case**
- **For delta-tracking, the actual value of the maximum cross-section over the interval was used, rather than an arbitrary guess.**
- **For the substepping method, equal-thickness subdivisions of the slab were used, with the number of subdivisions determined by trial and error to be the minimum required to match the accuracy of the other two methods**
- **1,000,000 histories were followed for each method in each of the test cases. Results are given in Table (2).**
- **Accuracy of all 3 methods is comparable, given that sufficient substeps are used for the substepping method. The number of collisions and the transmission at the right slab boundary are the same within statistics.**

## Compare: Direct Numerical, Delta-tracking, & Substepping

**Table 2**

Method	Collisions	Transmission	Function Evaluations per Collision
<b>1. Constant Cross-section</b>			
Substep	865001	0.1350	2.16
Delta-tracking	865362	0.1354	1.00
Direct	864513	0.1355	1.00
<b>2. Linearly Decreasing</b>			
Substep	865189	0.1348	29.90
Delta-tracking	865362	0.1346	1.48
Direct	865145	0.1349	2.95
<b>3. Linearly Increasing</b>			
Substep	864742	0.1353	29.92
Delta-tracking	864754	0.1352	2.77
Direct	864998	0.1350	4.38
<b>4. Exponentially Decreasing</b>			
Substep	282293	0.7177	107.08
Delta-tracking	283236	0.7168	5.37
Direct	282881	0.7171	3.78

## Compare: Direct Numerical, Delta-tracking, & Substepping

<b>5. Exponentially Increasing</b>			
Substep	931436	0.0686	33.20
Delta-tracking	931279	0.0687	7.55
Direct	931884	0.0681	4.94
<b>6. Sharp Gaussian</b>			
Substep	864591	0.1354	41.17
Delta-tracking	864494	0.1355	20.53
Direct	864518	0.1355	4.06
<b>7. Broad Gaussian</b>			
Substep	745446	0.2546	27.84
Delta-tracking	745301	0.2547	1.18
Direct	744956	0.2550	3.26

- **Function Evaluations per Collision:**
  - average number of flights per collision for the substepping method
  - average number of pseudocollisions (delta + real) for each real collision for the delta-tracking method
  - average number of Newton iterations per collision for the direct numerical method.
- **Both delta-tracking and the direct method are significantly more effective than substepping**
- **Delta-tracking and the direct method are roughly comparable, with delta-tracking being faster when there is little variation in the cross-section and the direct method being faster when there is more variation in the cross-section.**
- **Direct method should be viewed as an alternative to delta-tracking if there are large variations in cross-section.**

# Varying Material Properties

---

- **Represent material density by high-order, orthogonal polynomial expansion within each cell**
  - Legendre polynomial representation for material density in cell

$$\rho(x) = \sum_{n=0}^N \frac{2n+1}{2} \cdot a_n \cdot P_n \left[ \frac{2}{\Delta x} (x - x_{\min}) - 1 \right]$$

$$a_n = \frac{2}{\Delta x} \int_{x_{\min}}^{x_{\max}} \rho(x) P_n \left[ \frac{2}{\Delta x} (x - x_{\min}) - 1 \right] dx$$

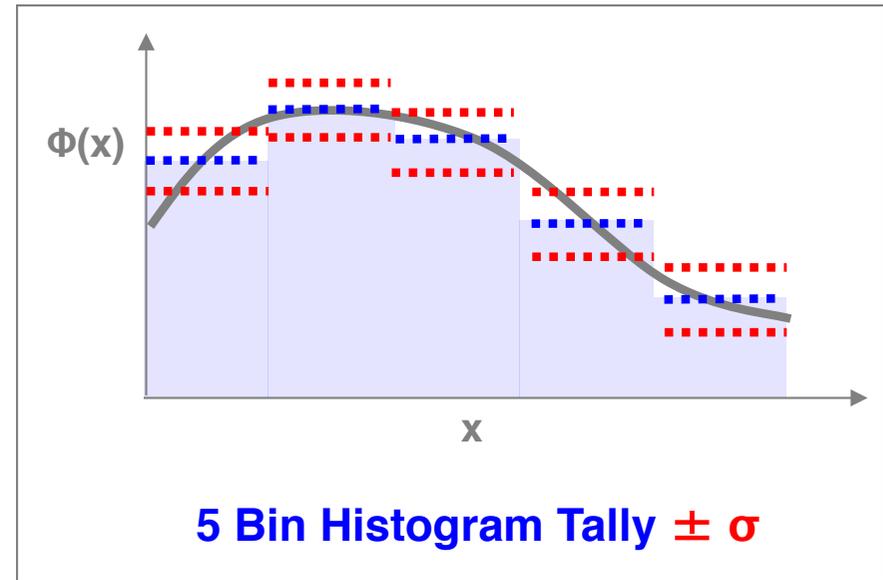
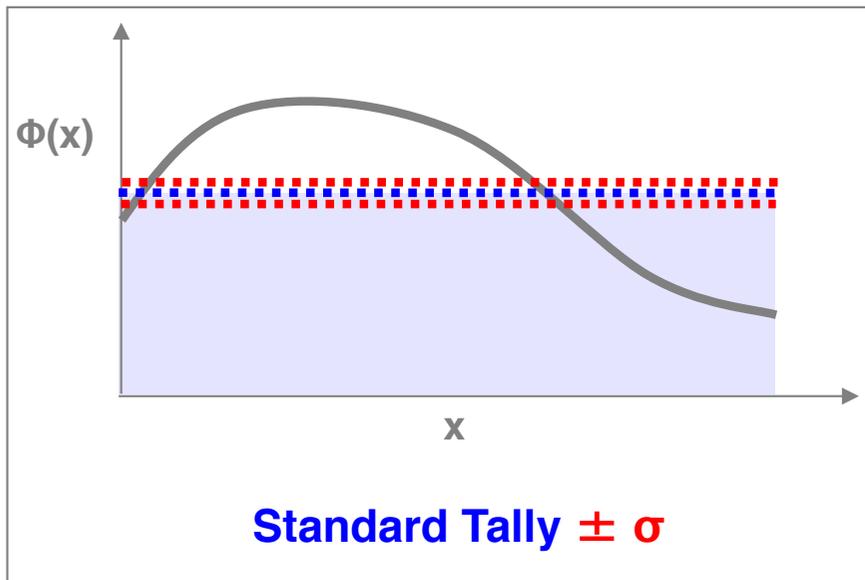
- **Sample the free-flight distance to next interaction using a direct numerical sampling scheme (Brown & Martin)**

$$\Sigma(x) = \frac{\rho(x)}{\rho_0} \cdot \Sigma_0, \quad \tau(s) = \frac{\Sigma_0}{\rho_0} \cdot \int_x^{x+s} \rho(x') \frac{dx'}{\mu}$$

- Use Newton iteration to solve nonlinear equation for flight path

# Continuously Varying Tallies

- **Conventional Monte Carlo codes tally integral results**
  - Tallies summed into bins
  - Zero-th order quantities within each bin
  - Stepwise approximation to results



- Unfortunately, by dividing the tally into bins we increase the variance of the estimate because relatively few histories score in an individual bin.

# Continuously Varying Tallies

---

- **An alternative to the histogram tally is the Functional Expansion Tally (FET).**
  - Tally the zeroth spatial moment of flux in each cell, AND higher moments with respect to some set of basis functions
  - Moments can then be used for a functional expansion of the flux distribution within the tally region
  
- **FET, Higher order tallies**
  - Represent results by high-order, orthogonal polynomial expansion within each cell
  - Make tallies for expansion coefficients
  - Legendre polynomial representation for continuous tallies

$$\Phi(x) = \sum_{n=0}^N \frac{2n+1}{2} \cdot b_n \cdot P_n \left[ \frac{2}{\Delta x} (x - x_{\min}) - 1 \right]$$

$$b_n = \frac{2}{\Delta x} \int_{x_{\min}}^{x_{\max}} \Phi(x) P_n \left[ \frac{2}{\Delta x} (x - x_{\min}) - 1 \right] dx$$

# Continuously Varying Tallies

---

- **Make tallies for the Legendre coefficients at each collision or flight:**

$$b_n = \frac{2}{\Delta x} \int_{x_{\min}}^{x_{\max}} \Phi(x) P_n \left[ \frac{2}{\Delta x} (x - x_{\min}) - 1 \right] dx$$

- **At collisions, tally**  $\frac{wgt}{\Sigma_T} \cdot P_n \left[ \frac{2}{\Delta x} (x - x_{\min}) - 1 \right]$  **for n=1..N**

- **At flights, tally**  $wgt \cdot \frac{1}{\mu} \int_x^{x+s} P_n \left[ \frac{2}{\Delta x} (x' - x_{\min}) - 1 \right] dx'$  **for n=1,N**

- **Reconstruct  $\Phi(x)$  and  $\sigma_\phi^2(x)$  from tallied coefficients**

# Continuously Varying Tallies

---

- FETs will have some amount of statistical uncertainty
- FET uncertainty in any expansion coefficient can be estimated with the sample variance statistic:

$$\hat{\sigma}_{\hat{a}_n}^2 = \frac{\sum_{i=1}^N \left( \sum_{c=1}^{C_i} w_{i,c} \psi_n(x_i) \right)^2 - \frac{1}{N} (\hat{a}_n)^2}{N(N-1)}$$

- It is also possible to derive a pointwise estimate of the variance in the functional approximation itself.

$$\hat{\sigma}_{\hat{\phi}}^2(x) = \frac{N}{N-1} \sum_{n=0}^M \sum_{m=0}^M \hat{\sigma}_{a_n a_m} k_n k_m \psi_n(x) \psi_m(x)$$

Sample covariance between coefficients  $\hat{a}_n$  and  $\hat{a}_m$

# Numerical Examples

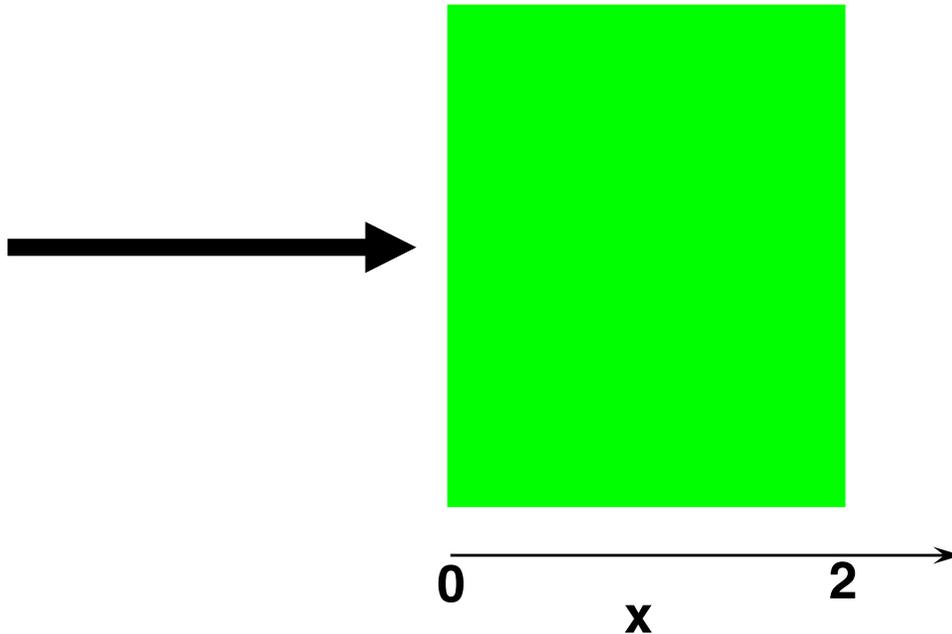
---

- **Demonstration Monte Carlo code**
  - 1D slab geometry
  - Piecewise Legendre expansion for material properties in each cell
  - Piecewise Legendre expansion for pathlength tallies within each cell
  - 5th order Legendre expansions, trivial to go higher
- **Examples**
  - A: Fixed source, beam into slab
  - B: Criticality, reflected reactor
- **Procedure**
  - Calculations with continuous materials + continuous tallies
  - Calculations with stepwise approximations: 2, 4, 8, 16, 32

# Varying Materials & Tallies - Example A

---

- **Beam source into slab**
  - Vacuum boundaries
  - Density in slab varies from 0 at edges to 10 at center
  - $\Sigma_T = 1.00$ ,  $\Sigma_s = 0.99$ ,  $\Sigma_A = 0.01$



# Varying Density - Problem A

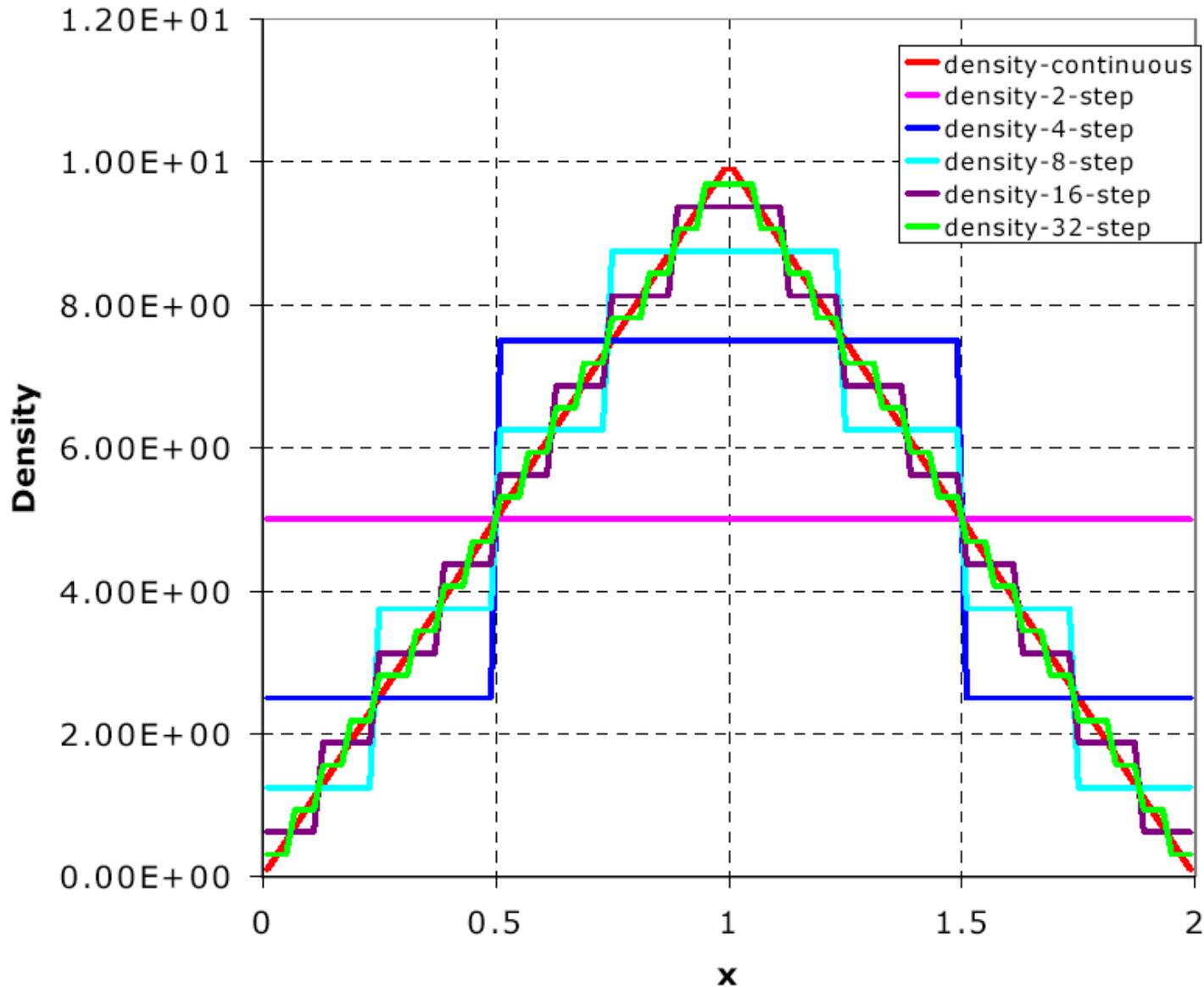


Figure 1. Density variation for Problem A: Continuous density and various stepwise approximations

# Continuous Tallies - Problem A

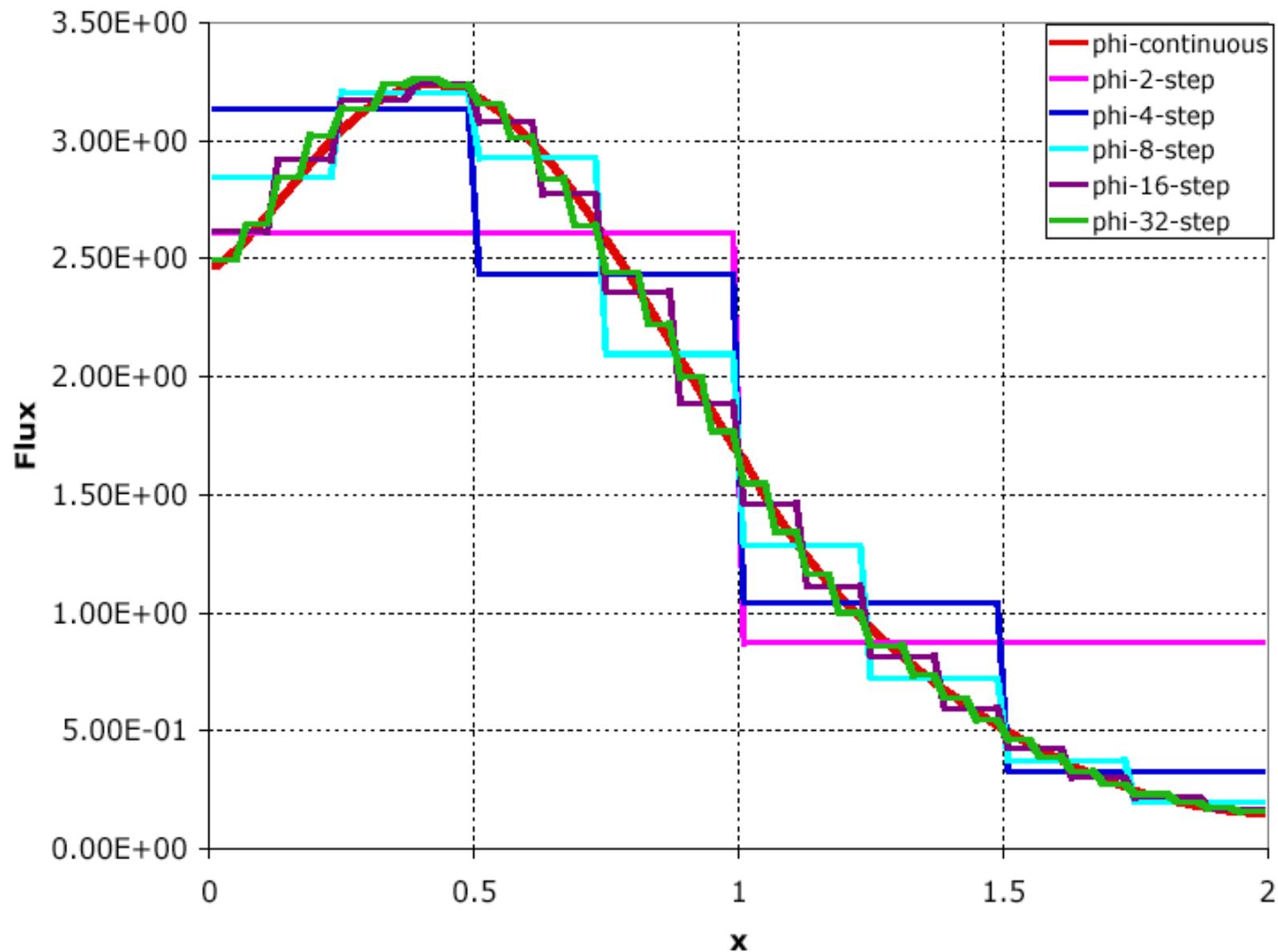


Figure 3. Flux results for Problem A: Beam into slab with linearly-varying density, continuous tally and various stepwise approximations

# Varying Materials & Tallies - Example B

---

- **Eigenvalue calculation - depleted core with reflector**

- Density varies quadratically in core: .25 at center, 2.25 at edges
- Constant density in reflector, 1.0
- Core:  $\Sigma_T = 2.00$ ,  $\Sigma_s = 0.125$ ,  $\Sigma_A = 1.025$ ,  $\Sigma_F = 0.85$ ,  $\nu = 2.4$
- Reflector:  $\Sigma_T = 0.25$ ,  $\Sigma_s = 0.24$ ,  $\Sigma_A = 0.01$



-2 -1.75

x

1.75 2

# Varying Density - Problem B

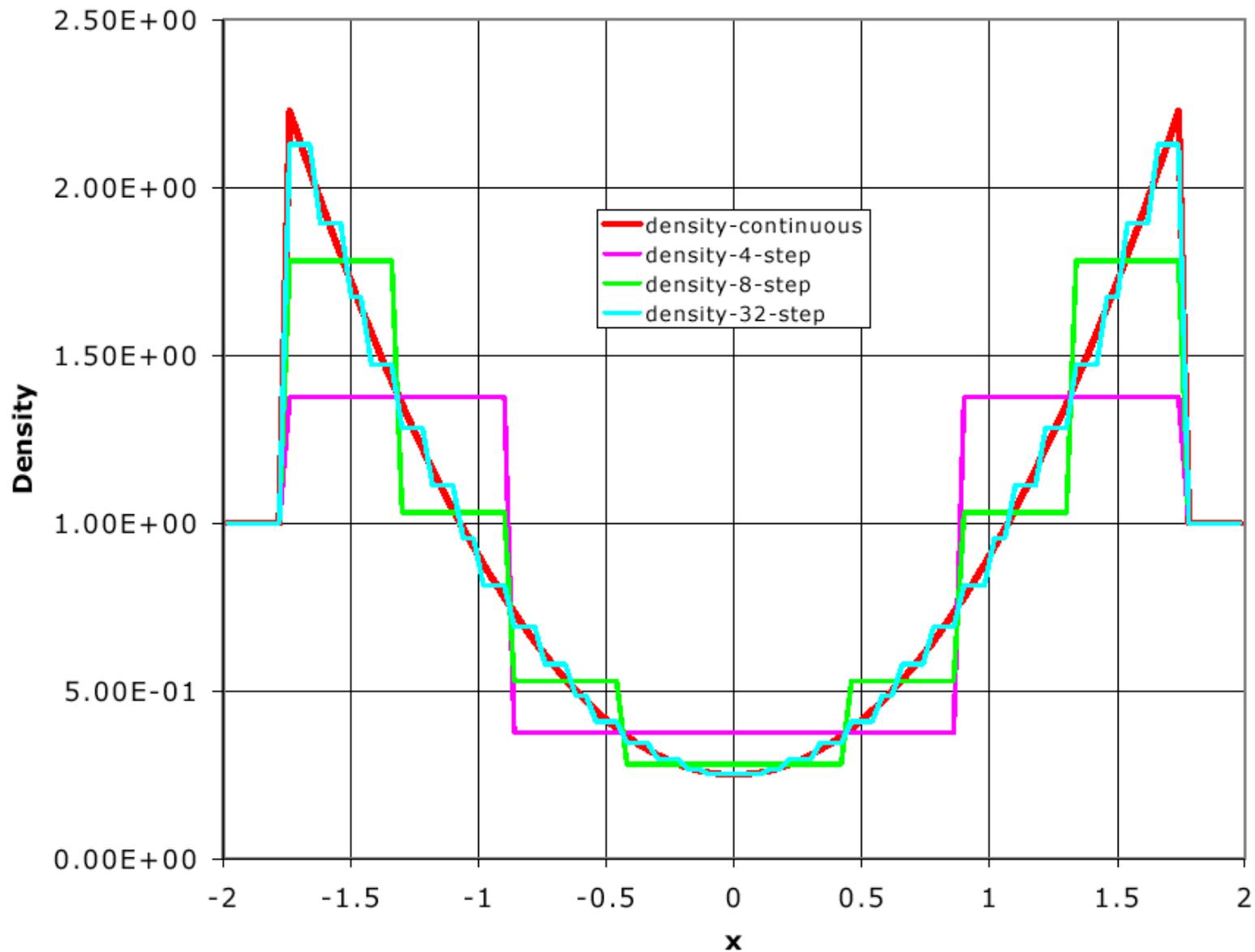


Figure 2. Density variation for Problem B: Quadratic variation for center fissionable region and various stepwise approximations

# Continuous Tallies - Problem B

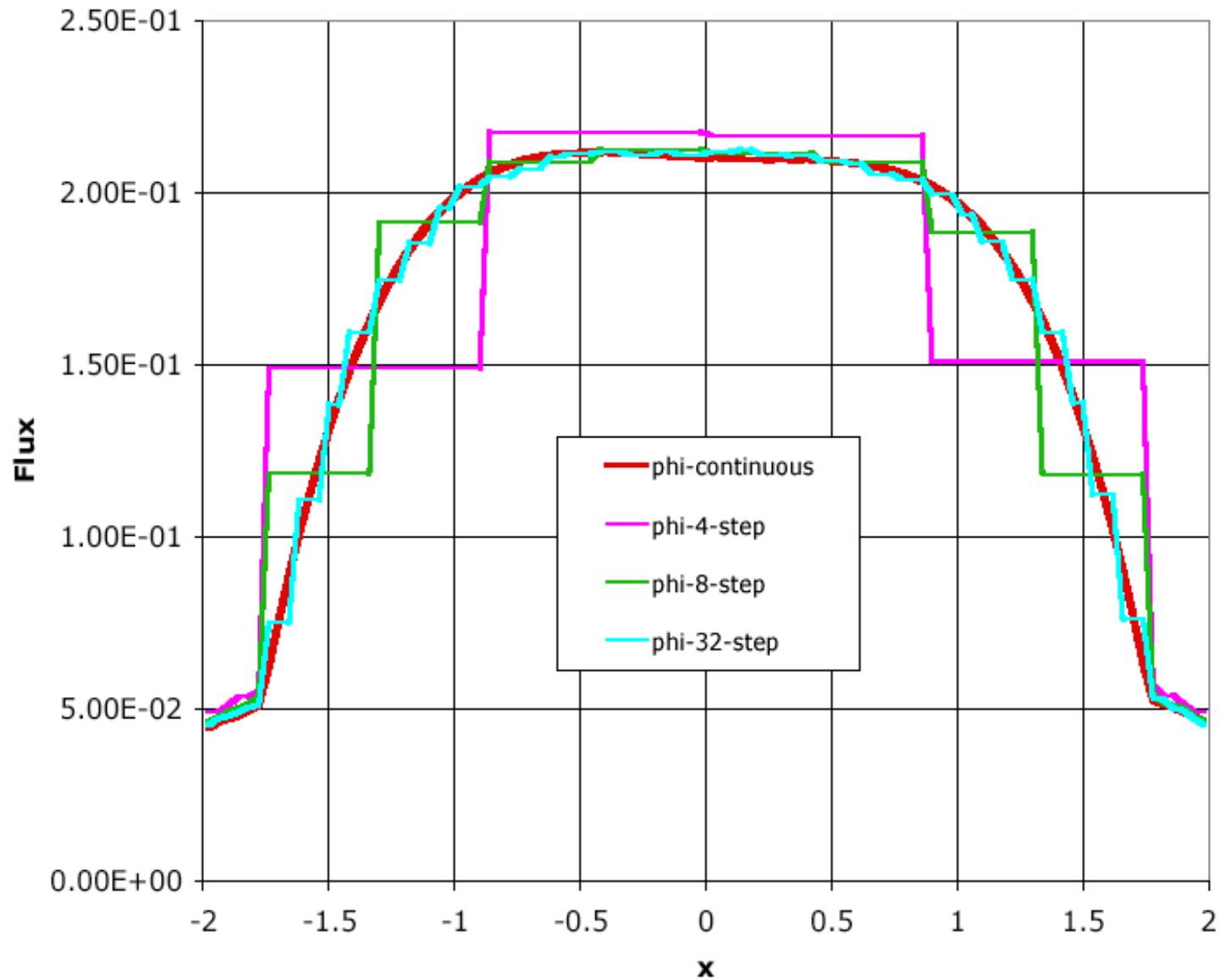


Figure 4. Flux results for Problem B: Quadratic density variation, continuous tallies and various stepwise approximations

# Continuous Materials & Tallies

---

- **Timings for Problem B**

Continuous	(3 cells)	102 sec
4-step	(12 cells)	117 sec
8-step	(24 cells)	130 sec
32-step	(96 cells)	283 sec

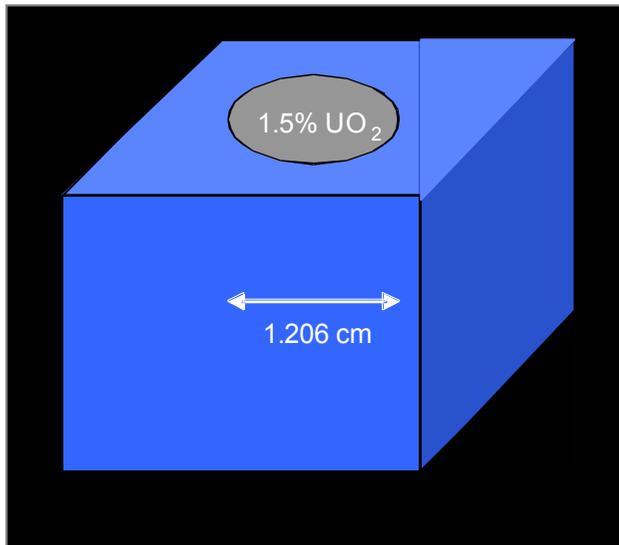
**Continuous tallies require more work, but fewer cells.  
Can give computational advantage for some problems.**

- **Conclusions**

- It is now practical to extend Monte Carlo codes to use continuously varying material properties & tallies
- 5th order Legendre polynomials within each cell look promising

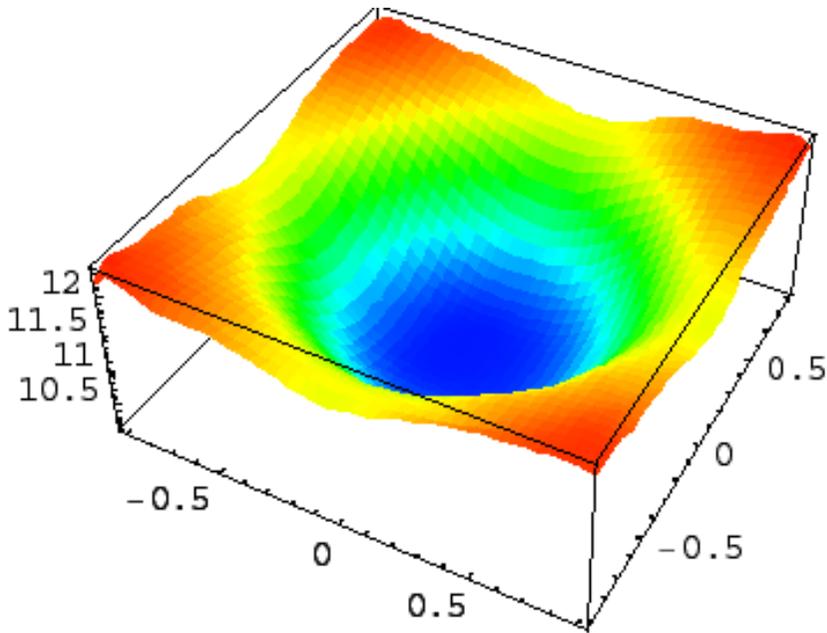
# Continuous 2D Tallies - Reactor Fuel Pin Cell

- For testing the 2-D FET was implemented in MCNP4c.
- Benchmark tests were conducted on a simulated PWR fuel pin to calculate the spatial distribution of thermal flux in the x-y plane.
- The FET results were compared with results from an MCNP5 mesh tally calculation.

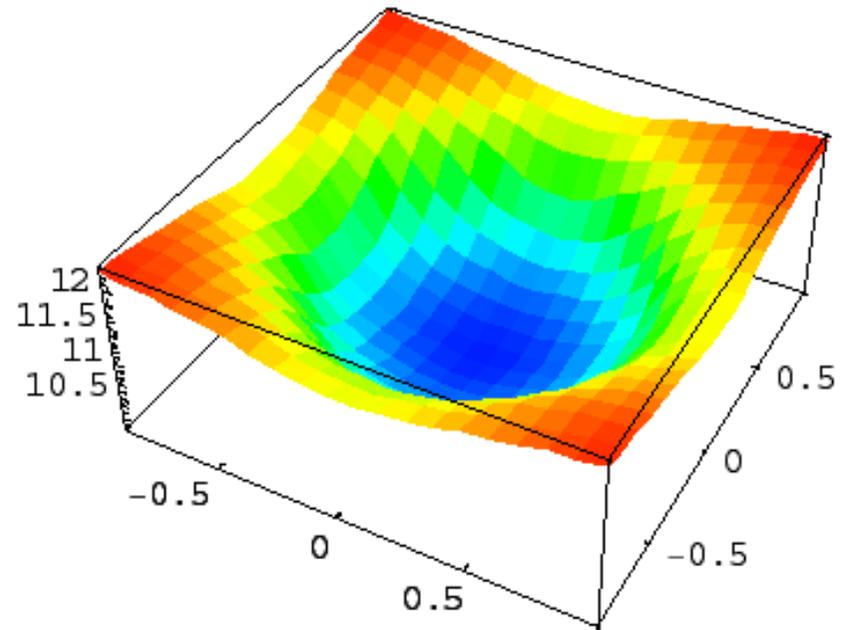


- Fuel OD: 1.206 cm
- Pitch: 1.875 cm
- Clad Thickness: 0.06 cm
- Gap Thickness: 0.008 cm
- Fuel Enrichment: 1%
- Eigenvalue: 1.026

# Continuous 2D Tallies - Reactor Fuel Pin Cell



**Figure 2a.** 9x9 Legendre expansion tally for thermal neutron flux across the fuel pin obtained in a 2 million history simulation.

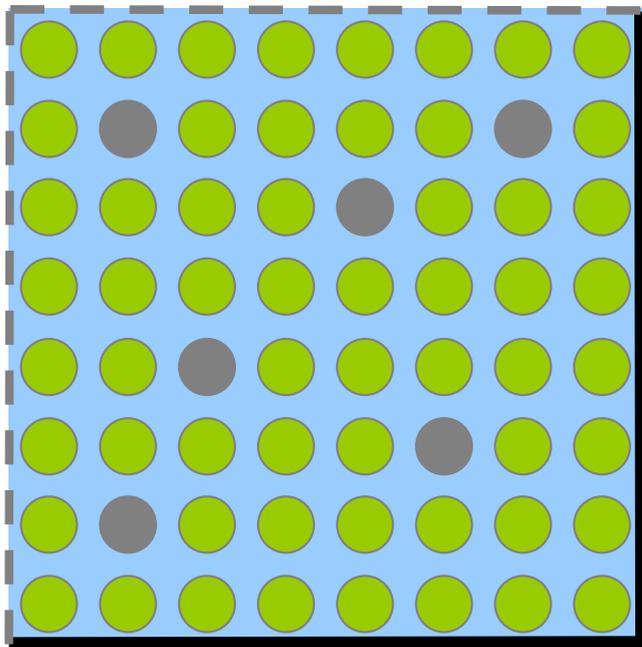


**Figure 2b.** MCNP5 20x20 mesh tally for thermal neutron flux across the fuel pin obtained in a 2 million history simulation.

# Continuous 2D Tallies - 1/4 Fuel Assembly

---

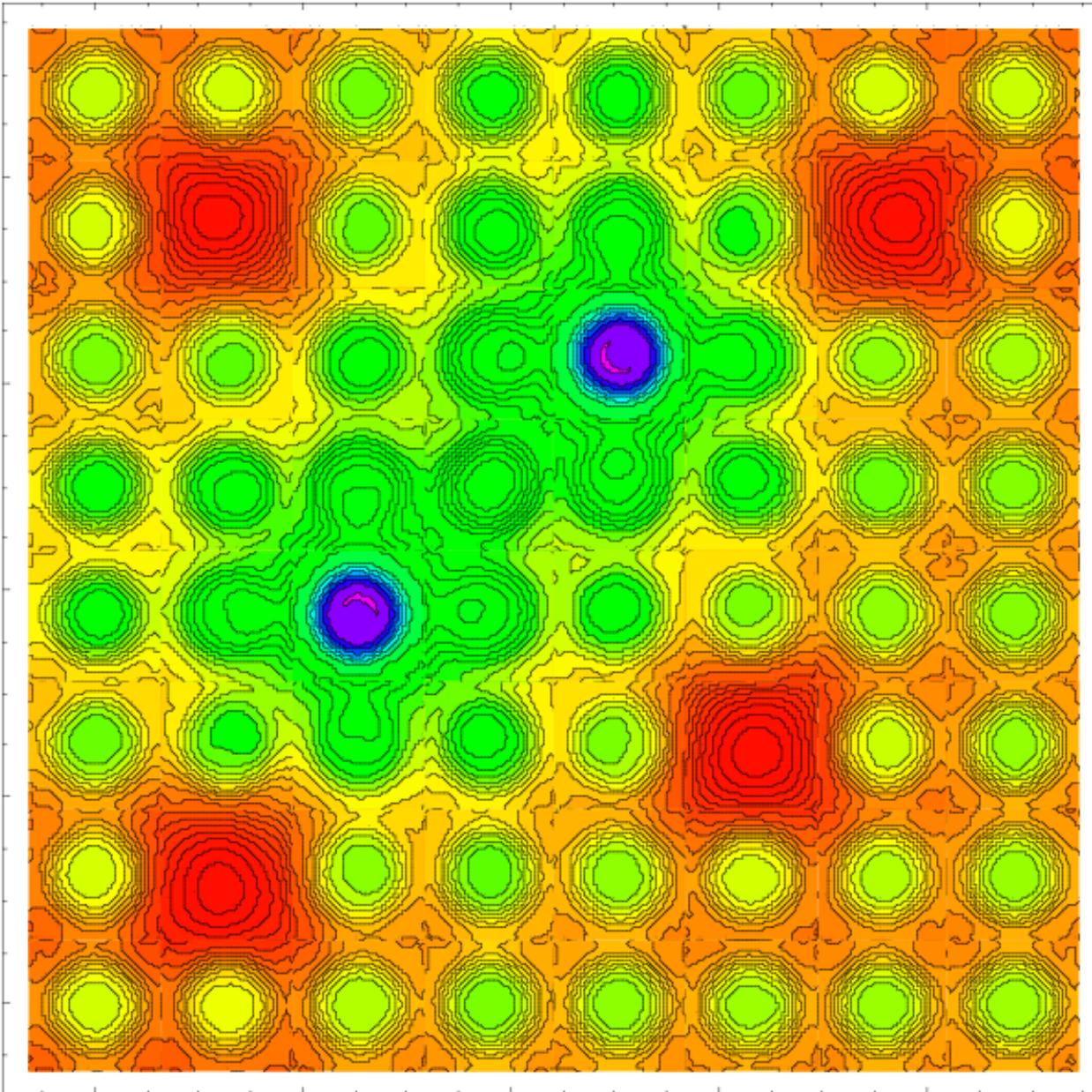
- In order to test the method on a more realistic problem a PWR-type quarter assembly was modeled.
- For testing the 2-D FET was implemented in MCNP4c.
- A separate 2-D Legendre expansion was used for each individual pin-cell.



- 1/4 of a 16 × 16 fuel assembly
- 58 fuel pins, 4 control rods, 2 burnable poison pins
- Pitch: 1.26
- Fuel Enrichment: 2%
- Burnable Poison Density: 0.30 g/cc
- Eigenvalue: 1.1709

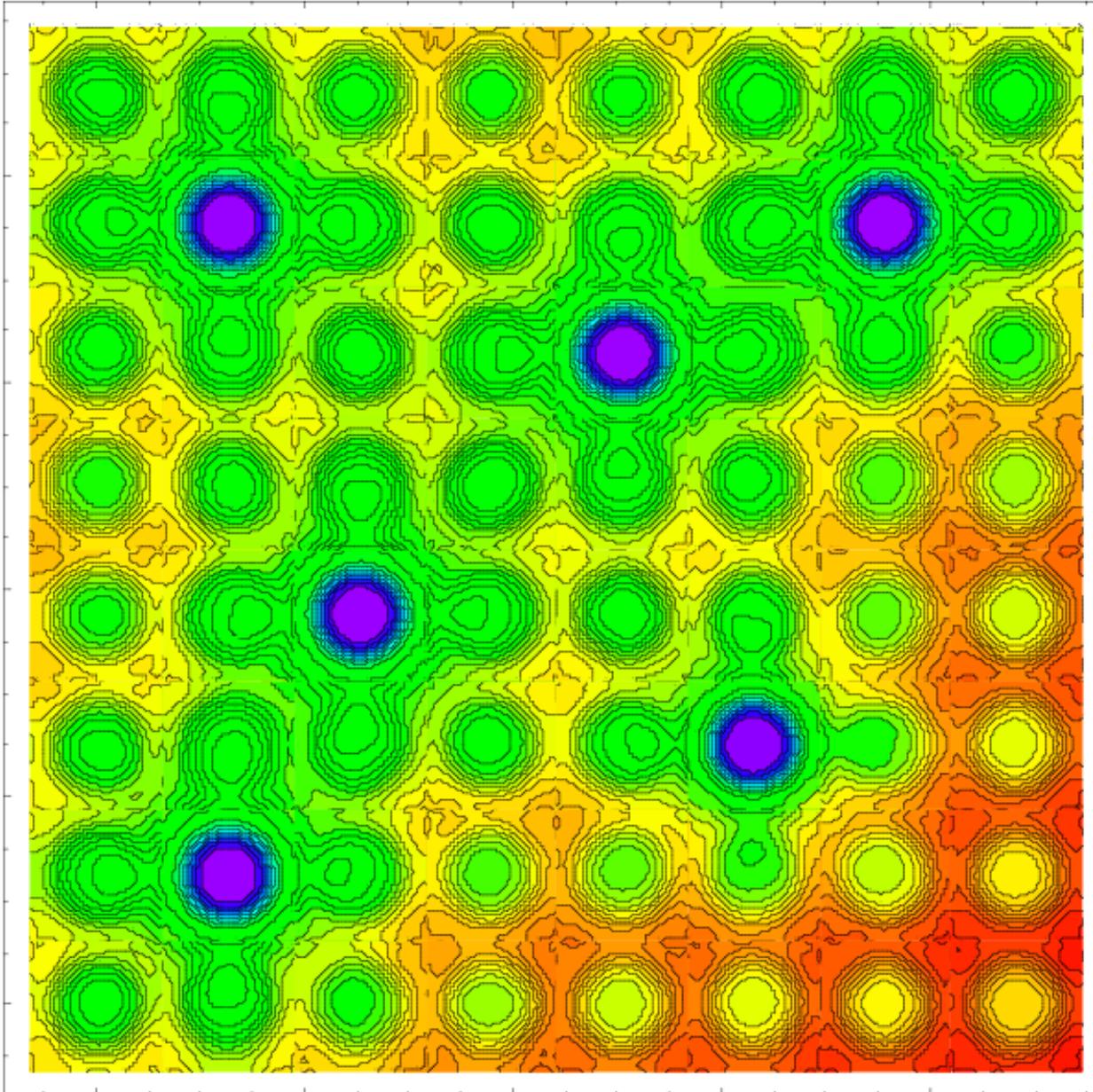
# Continuous 2D Tallies - 1/4 Fuel Assembly

- Control rods withdrawn



# Continuous 2D Tallies - 1/4 Fuel Assembly

- Control rods inserted



# Conclusions

---

- **It is now practical to extend Monte Carlo codes to use continuously varying material properties & tallies**
  - Continuous materials can be modeled with delta-tracking or direct numerical tracking
  - Flux expansion tallies have many benefits over traditional histogram tallies
    - Can obtain a functional form for the tally distribution directly from the Monte Carlo, with no post-processing or curve fitting required.
    - Can provide a more accurate approximation than a histogram tally, without requiring significantly more work per history.
    - FET estimators are easy to implement in existing Monte Carlo codes
- **Future work:**
  - Practical implementation in production Monte Carlo codes
    - See Griesheimer & Martin PHYSOR-2004 paper for 2D extension
    - Will be added to MCNP5 mesh tallies in near future
  - Investigate methods for reactor depletion

# References - Continuous Materials & Tallies

---

- FB Brown, D Griesheimer, & WR Martin, "Continuously Varying Material Properties and Tallies for Monte Carlo Calculations", PHYSOR-2004, Chicago, IL (April, 2004)
- FB Brown & WR Martin, "Direct Sampling of Monte Carlo Flight Paths in Media with Continuously Varying Cross-sections", ANS Mathematics & Computation Topical Meeting, Gatlinburg, TN (April, 2003).
- W.R. Martin and F.B. Brown, "Comparison of Monte Carlo Methods for Nonlinear Radiation Transport," *Proc. ANS Mathematics and Computations Topical Meeting*, Salt Lake City (Sept 2001)
- DP Griesheimer & WR Martin, "Estimating the Global Scalar Flux Distribution with Orthogonal Basis Function Expansions", *Trans. Am. Nucl. Soc.* 89 (Nov, 2003)
- DP Griesheimer & WR Martin, "Two Dimensional Functional Expansion Tallies for Monte Carlo Simulations," PHYSOR-2004, Chicago, IL (April, 2004)
- ER Woodcock, T Murphy, PJ Hemmings, TC Longworth, "Techniques Used in the GEM Code for Monte Carlo Neutronics Calculations in Reactors and Other Systems of Complex Geometry," *Proc. Conf. Applications of Computing Methods to Reactor Problems*, ANL-7050, p. 557, Argonne National Laboratory (1965).
- LL Carter, ED Cashwell, & WM Taylor, "Monte Carlo Sampling with Continuously Varying Cross Sections Along Flight Paths", *Nucl. Sci. Eng.* 48, 403-411 (1972).
- J. Spanier, "Monte Carlo Methods for Flux Expansion Solutions of Transport Problems," *Nucl. Sci. Eng.*, 133, 73 (1999).
- CJ Everett, ED Cashwell, RG Schrandt, "A Monte Carlo Transport Routine for the 'US Standard Atmosphere' (1962) to an Altitude of 90 Kilometers," LA-5089-MS, Los Alamos National Laboratory (1972).
- G.E. Thomas and K. Stamnes, Radiative Transfer in the Atmosphere and Oceans, p. 429, Cambridge University Press (1999)
- H.D. Rees, *Phys. Lett. A* 26, 416 (1968), also *J. Phys. Chem. Solids* 30, 643 (1969).
- M.M.R. Williams, Random Processes in Nuclear Reactors, Pergamon Press (1974).
- S.N. Cramer, "Application of the Fictitious Scattering Radiation Transport Model for Deep-Penetration Monte Carlo Calculations," ORNL/TM-4880, Oak Ridge National Laboratory (1977).
- W.A. Coleman, "Mathematical Verification of a Certain Monte Carlo Sampling Technique and Applications of the Technique to Radiation Transport Problems," *Nucl. Sci. Eng.* 32, 76-81 (1968).







**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Random Number Generators for Particle Transport MC & RNG Testing**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Outline

---

- **Random Number Generators for Particle Transport MC**
- **Random Number Generator Testing**

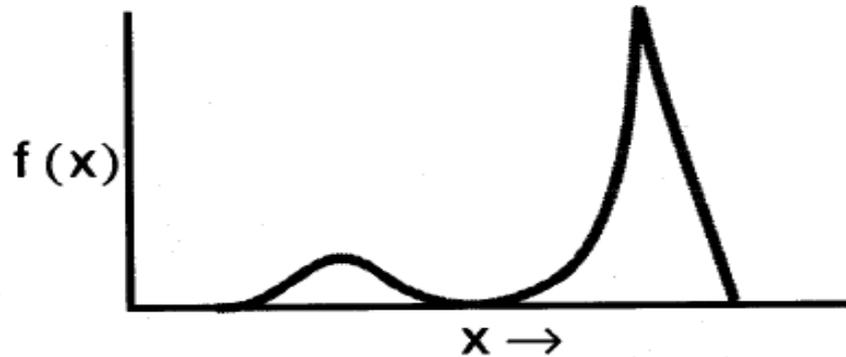
# Introduction

---

The key to Monte Carlo methods is the notion of *random sampling*.

- The problem can be stated this way:

Given a probability density,  $f(x)$ , produce a sequence of  $\hat{X}$ 's.  
The  $\hat{X}$ 's should be distributed in the same manner as  $f(x)$ .



- Random sampling distinguishes Monte Carlo from other methods
- When Monte Carlo is used to solve the Boltzmann transport equation:
  - Random sampling models the outcome of physical events (e.g., neutron collisions, fission process, sources, .....

# Monte Carlo & Random Sampling

---

## Categories of random sampling

- **Random number generator** → uniform PDF on (0,1)
- Sampling from **analytic PDFs** → normal, exponential, Maxwellian, ...
- Sampling from **tabulated PDFs** → angular PDFs, spectrum, ...

## For Monte Carlo codes...

- **Random numbers,  $\xi$ , are produced by the RN generator on (0,1)**
- **Non-uniform random variates are produced from the  $\xi$ 's by:**
  - Direct inversion
  - Rejection methods
  - Transformations
  - Composition (mixtures)
  - Sums, products, ratios, ...
  - Table lookup + interpolation
  - Lots (!) of other tricks
- **Typically < 5 - 10% of total CPU time**

---

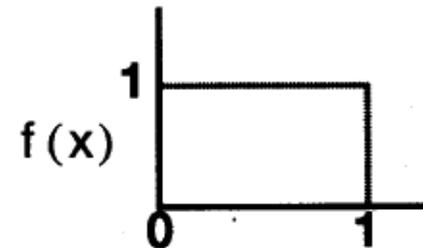
# Random Number Generators For Particle Transport MC

# Random Number Generators

---

"Randomness is a negative property; it is the absence of any pattern."  
Richard W. Hamming, 1991

- Numbers are not random; a sequence of numbers can be.
- Truly random sequences are generally not desired on a computer.
- **RNG**
  - Function which generates a sequence of numbers which appear to have been randomly sampled from a uniform distribution on  $(0,1)$
  - Repeatable (deterministic)
  - Pass statistical tests for randomness
  - Typical usage in codes:  **$r = \text{rang}()$**
  - Also called "pseudo-random number generators"
- All other random sampling is performed using this basic RNG
- Note that the probability of something occurring also varies in  $(0,1)$  between 0 & 1 .....



# RNGs – Some Comments . . . . .

---

## From “An Essay on Random Number Generators for Monte Carlo Codes”, F. Brown:

Numbers are not random, they are just numbers.

An algorithm for producing "random numbers" is not random, it is a fully prescribed sequence of operations to be performed.

When we talk about "random numbers" on a computer, what we really mean is "a sequence of numbers that appears to be uniformly distributed". Similarly, a "random number generator" is an algorithm for producing a sequence of numbers that appears to be uniformly distributed.

We can't determine whether a single number was produced "randomly". We can, however, subject a sequence (or stream) of numbers to a set of statistical tests. We then compare the outcome of these tests to the known theoretical results that would be produced if a truly random, uniformly distributed sequence of numbers was subjected to the same tests. If our algorithm-produced sequence yields the same results as a theoretical truly random sequence for all of the tests, we declare that our algorithm-produced sequence is "random". What we mean is that the algorithm-produced sequence is indistinguishable from a truly random sequence.

In considering a random number generator for Monte Carlo codes, we are fully aware that algorithm-produced sequences are deterministic, not truly random. However, if a sequence of algorithm-produced random numbers is indistinguishable from a truly random sequence, then we may confidently use it in Monte Carlo simulations.

An important theme of the preceding discussion is that we cannot prove that a sequence is random, nor that a given random number generator produces a random sequence. All we can do is subject the algorithm-produced sequences to a set of statistical tests and compare to theoretical results for truly random sequences. If an algorithm-produced sequence fails any of the tests, then we can declare that the random number generator is bad. Hence, given a comprehensive set of statistical tests, we can identify bad generators, but cannot prove that a generator is good.

# RNGs for Particle Transport MC – Needed Properties

---

- **Quality - depends on what it is to be used for:**
  - **Cryptography**
    - Every bit in every integer in a sequence must be random & unpredictable
  - **Particle transport**
    - We convert a random sequence of integers into a sequence of floating-point (real) numbers. Not concerned about the last few least-significant bits
- **Reproducibility**
  - **For any combination of the number of processors, MPI tasks, threads, or spatial domains (for domain decomposition):**
    - Want same results (ie, RN usage for each particle)
    - The order in which particles are processed should not affect results
    - Every particle created must be given a unique RN seed
- **Skip-Ahead**
  - **For parallel calculations, must be a fast way to skip-ahead in the RN sequence**
- **State**
  - **The RNG state-space or storage per particle must be small**
- **Robust**
  - **Must never, ever fail !!!**

# Linear Congruential RNGs (LCGs)

---

Most production-level Monte Carlo codes for particle transport use linear congruential random number generators (LCGs):

$$S_{i+1} = S_i \cdot g + c \quad \text{mod } 2^m$$

$S_i$  = seed,  $g$  = multiplier,  $c$  = adder,  $2^m$  = modulus

- Simple, fast, robust, over 60 years of heavy-duty use
- Theory is well-understood (e.g., DE Knuth, Vol. 2, 177 pages)
- Not the "best" RNGs, but good enough - RN's are used in unpredictable ways during particle simulation
- To achieve reproducibility for vector or parallel calculation, there must be a fast, direct method for **skipping ahead** (or back) in the random sequence

# Simple LCG - Example #1

---

$$S_{k+1} = [ g \cdot S_k + C ] \text{ mod } p, \quad \text{with } g=47, C=1, S_0=1, P=100$$

$$S_{k+1} = [ 47 \cdot S_k + 1 ] \text{ mod } 100$$

```
s( 0) = 1
s( 1) = (47x1  + 1) mod 100 =  48 mod 100 = 48
s( 2) = (47x48 + 1) mod 100 = 2257 mod 100 = 57
s( 3) = (47x57 + 1) mod 100 = 2680 mod 100 = 80
s( 4) = (47x80 + 1) mod 100 = 3761 mod 100 = 61
s( 5) = (47x61 + 1) mod 100 = 2868 mod 100 = 68
s( 6) = (47x68 + 1) mod 100 = 3197 mod 100 = 97
s( 7) = (47x97 + 1) mod 100 = 4560 mod 100 = 60
s( 8) = (47x60 + 1) mod 100 = 2821 mod 100 = 21
s( 9) = (47x21 + 1) mod 100 =  988 mod 100 = 88
s(10) = (47x88 + 1) mod 100 = 4137 mod 100 = 37
s(11) = (47x37 + 1) mod 100 = 1740 mod 100 = 40
s(12) = (47x40 + 1) mod 100 = 1881 mod 100 = 81
s(13) = (47x81 + 1) mod 100 = 3808 mod 100 =  8
s(14) = (47x8  + 1) mod 100 =  377 mod 100 = 77
s(15) = (47x77 + 1) mod 100 = 3620 mod 100 = 20
s(16) = (47x20 + 1) mod 100 =  941 mod 100 = 41
s(17) = (47x41 + 1) mod 100 = 1928 mod 100 = 28
s(18) = (47x28 + 1) mod 100 = 1317 mod 100 = 17
s(19) = (47x17 + 1) mod 100 =  800 mod 100 =  0
s(20) = (47x0  + 1) mod 100 =    1 mod 100 =  1
s(21) = (47x1  + 1) mod 100 =  48 mod 100 = 48
s(22) = (47x48 + 1) mod 100 = 2257 mod 100 = 57
```

# Simple LCGs - Examples #2 & #3

---

**Example #2:**  $S_{k+1} = [ 5 \cdot S_k + 1 ] \text{ mod } 100,$

$$s(0) = 1$$

$$s(1) = (5 \times 1 + 1) \text{ mod } 100 = 6 \text{ mod } 100 = 6$$

$$s(2) = (5 \times 6 + 1) \text{ mod } 100 = 31 \text{ mod } 100 = 31$$

$$s(3) = (5 \times 31 + 1) \text{ mod } 100 = 156 \text{ mod } 100 = 56$$

$$s(4) = (5 \times 56 + 1) \text{ mod } 100 = 281 \text{ mod } 100 = 81$$

$$s(5) = (5 \times 81 + 1) \text{ mod } 100 = 406 \text{ mod } 100 = 6$$

$$s(6) = (5 \times 6 + 1) \text{ mod } 100 = 31 \text{ mod } 100 = 31$$

etc.

**Example #3:**  $S_{k+1} = [ 5 \cdot S_k + 0 ] \text{ mod } 100,$

$$s(0) = 1$$

$$s(1) = (5 \times 1) \text{ mod } 100 = 5 \text{ mod } 100 = 5$$

$$s(2) = (5 \times 5) \text{ mod } 100 = 25 \text{ mod } 100 = 25$$

$$s(3) = (5 \times 25) \text{ mod } 100 = 125 \text{ mod } 100 = 25$$

$$s(4) = (5 \times 25) \text{ mod } 100 = 125 \text{ mod } 100 = 25$$

etc.

# Choosing Parameters for LCGs

$$s_{k+1} \leftarrow [g \cdot s_k + c] \bmod p$$

- Modulus (p):

- choose  $p = 2^N$
- simplifies "mod p"— discard all but the N least significant bits
- simplifies division by p— shift the "point" left by N bits
- N should be as large as possible,  $N > 35$  is best.
- Usually, choose N to be number of bits in largest positive integer.

- Generator (g), Initial Seed ( $s_0$ ), & Increment (c) :

- choose g & c to maximize the period
- large g is best to reduce serial correlation
- obviously,  $g=1$  or  $g=0$  are bad

See Knuth, Vol 2

- For  $c = 0$  (multiplicative PRNG):

- choosing
- (1)  $g \bmod 8 = 3$  or  $5$
  - (2)  $s_0 = \text{odd}$

results in:      period =  $2^{N-2}$ ,      the maximum possible period.

- For  $c > 0$  (mixed PRNG):

- choosing
- (1) c relatively prime to p
  - (2) (g-1) to be a multiple of every prime factor of p
  - (3) (g-1) to be a multiple of 4 if p is a multiple of 4

results in:      period =  $2^N$ ,      the maximum possible period.

# RNG Example (very old)

## Example -- CYBER-205 RANF

$$s_{k+1} \leftarrow [g \cdot s_k + c] \text{ mod } p$$

FORTRAN	META	
common /q8ranfc/ seed	LOD s_descr, s	*load the seed
r = ranf()	EX g, 84000335758957	*generator
	EX e, 65489	*exponent, 2**-47
	MPYL g, s, s	*mult, keep last 47 bits
	STO s_descr, s	*store new seed
	PACK e, s, r	*insert exponent
	ADDN r, , r	*normalized result

- Note:
- (1)  $0 < r < 1$
  - (2) scalar timing ~320 ns / prn
  - (3) to vectorize — "unroll" or "replicate", vector timing ~30 ns / m

## How long will the PRNs last ?

time to generate ALL  $2^{45}$  RNs

Sharp EL-515s		1 M yr
CYBER-205,	scalar	4 mos
CRAY-1,	vector	15 days
CYBER-205,	2-pipe vector	12 days
CYBER-205,	4-pipe vector	6 days
CRAY-XMP/48,	vector x 4	3 days
CRAY-2,	vector x 4	30 hr
ETA-10,	vector x 8	13 hr
cray-c90,	vector x 16	4 hr

# LCGs – Last Few Bits

---

## Aside ...

For the multiplicative congruential method,  
why is the period limited to a maximum of  $2^{N-2}$  ??

$$s_{k+1} \leftarrow g \cdot s_k \pmod{p}, \quad s_0 \text{ odd, } g \pmod{8} = 3 \text{ or } 5$$

- All  $s_k$ 's are odd,  $g$  is odd

$\Rightarrow g \cdot s_k$  will always be odd,

reduces period by a factor of two.

- For  $g \pmod{8} = 3$ , trailing bits of  $g$  are (...011)

$$g \cdot s_k = (...011) \cdot (...11) = (...11)$$

or

$$g \cdot s_k = (...011) \cdot (...01) = (...01)$$

$\Rightarrow$  next-to-last bit of  $s_k$  will not change, reduces period by a factor of two.

- For  $g \pmod{8} = 5$ , trailing bits of  $g$  are (...101)

$$g \cdot s_k = (...101) \cdot (...1x1) = (...1x1)$$

or

$$g \cdot s_k = (...101) \cdot (...0x1) = (...0x1)$$

$\Rightarrow$  third-to-last bit of  $s_k$  will not change, reduces period by a factor of two.

# Typical Linear Congruential RNGs

- Multiplicative congruential method - Lehmer

$$S_k = g \cdot S_{k-1} + c \pmod{2^m}, \quad 0 < S_k < 2^m, \text{ integer}$$

$$\xi_k = S_k / 2^m, \quad 0 \leq \xi_k < 1, \text{ real}$$

- Typical parameters

	$2^m$	Period	$g$	$c$
RACER (KAPL)	$2^{47}$	$2^{45}$	84,000,335,758,957	0
RCP (BAPL)	$2^{48}$	$2^{48}$	$2^9+1$ 59,482,192,516,946	
MORSE (ORNL)	$2^{47}$	$2^{45}$	$5^{15}$	0
MCNP (LANL)	$2^{48}$	$2^{46}$	$5^{19}$	0
VIM (ANL)	$2^{48}$	$2^{46}$	$5^{19}$	0
RANF (CRAY)	$2^{48}$	$2^{46}$	44,485,709,377,909	0
G. Marsaglia	$2^{32}$	$2^{32}$	69069	1
MCNP5 (LANL)	$2^{63}$	$2^{63}$	(7 options)	1 or 0
MCNP6 (LANL)	$2^{63}$	$2^{63}$	(7 options)	1 or 0

# MCNP5 & MCNP6 RNG

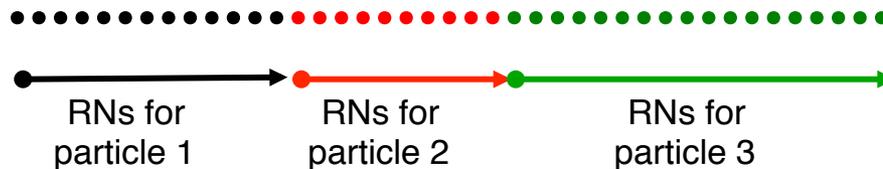
---

$$S_{n+1} = g S_n + c \pmod{2^m}$$

- See Knuth for rules for selecting  $g, c, m$  so that period is maximized & correlation minimized
- 7 different LCGs are available -- chosen based on the spectral test, Knuth's tests, & Marsaglia's DIEHARD tests
- LCG(  $g, c, m$  ):
  - **Traditional MCNP,** **period =  $2^{46} \approx 7 \times 10^{13}$**   
 #1 - LCG(  $5^{19}, 0, 48$  )
  - **L'Ecuyer 63-bit Mixed LCGs,** **period =  $2^{63} \approx 9 \times 10^{18}$**   
 #2 - LCG(  $9219741426499971445, 1, 63$  )  
 #3 - LCG(  $2806196910506780709, 1, 63$  )  
 #4 - LCG(  $3249286849523012805, 1, 63$  )
  - **L'Ecuyer 63-bit Multiplicative LCGs,** **period =  $2^{61} \approx 2 \times 10^{18}$**   
 #5 - LCG(  $3512401965023503517, 0, 63$  )  
 #6 - LCG(  $2444805353187672469, 0, 63$  )  
 #7 - LCG(  $1987591058829310733, 0, 63$  )

## Using RNGs in Particle Transport MC Codes

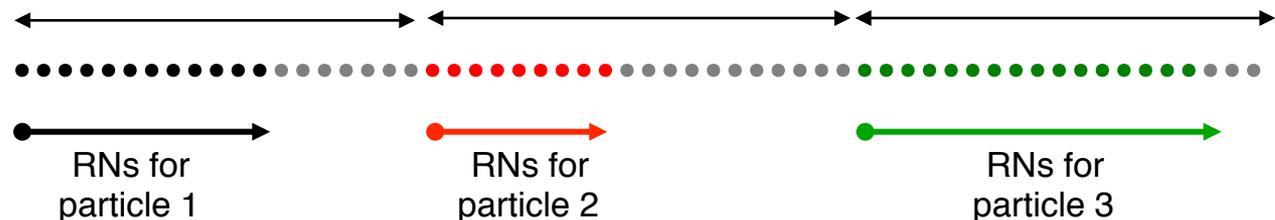
- Naïve use, in many older codes & student codes



- **Problem:** Can't start Particle-2 until Particle-1 is finished, etc.  
Can't do parallel processing of different particles

- MCNP, VIM, RACER, MC21, & many other production codes

- Partition RN sequence into equal-length subsequences, one for each particle

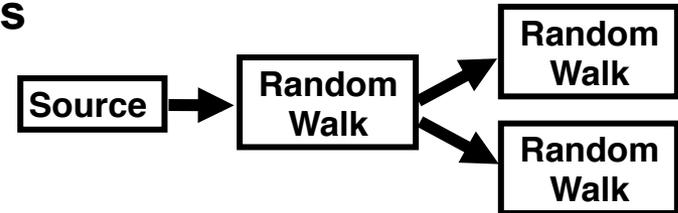


- Can process all particles in parallel
- Length of each subsequence is called the **stride**
- Must have a fast way to **skip-ahead** in the RN sequence

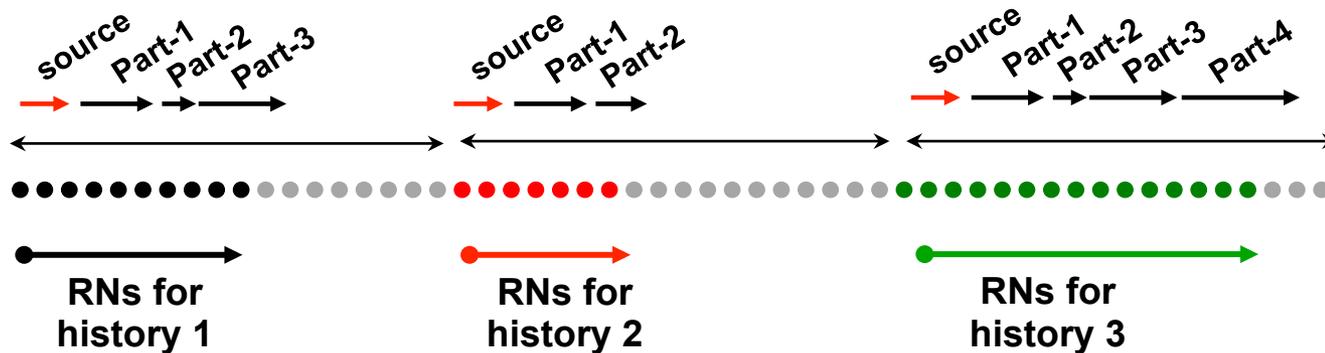
## Using RNGs in Particle Transport MC Codes

### • Histories vs particles

- With splitting &/or secondary particle creation, the number of particles in a given history is not known in advance



- Need to partition RN sequence by **history**, not by particle



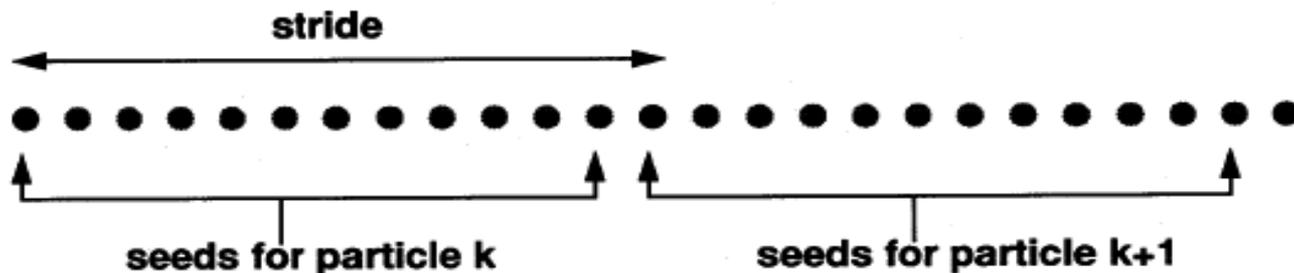
- With this scheme, can process histories in parallel, but not particles in same history
- Must have a predictable scheme for banking/unbanking particles in a given history (e.g., LIFO)

# Reproducibility

---

## Reproducibility of a Particle History

- use separate, distinct random sequence for each particle
- starting seeds for separate particles are separated by "stride"

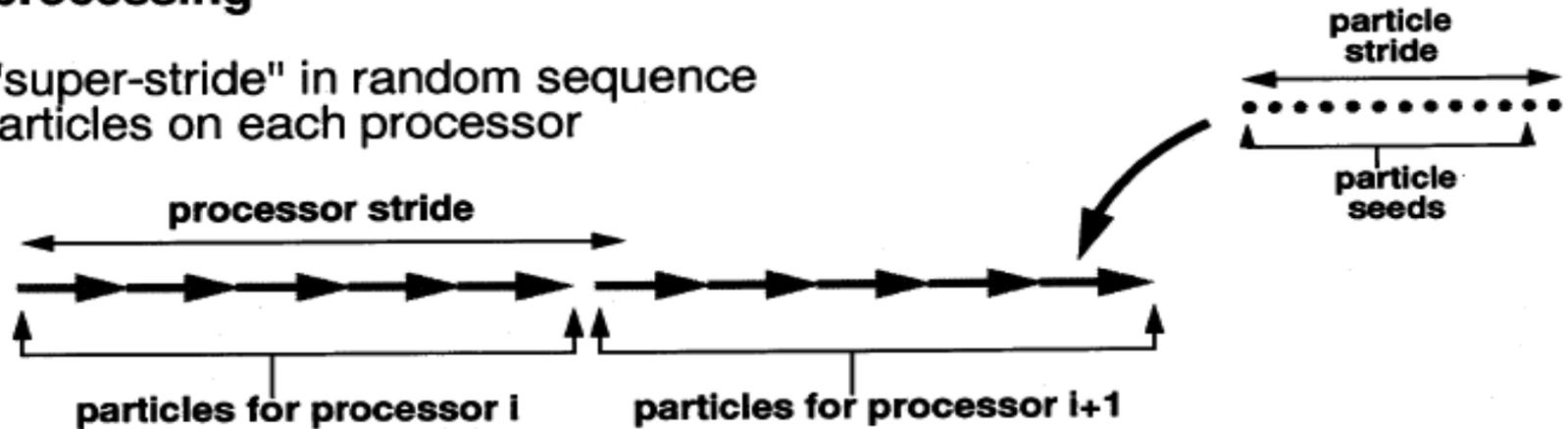


- stride should be large enough to prevent overlap (for most histories)
  - 1000 is common for reactor analysis problems
    - splitting & variance reduction not needed for in-core physics
    - reduces total random number usage
  - 4,297 is the "old" default for MCNP & VIM
  - 152,917 is the default for MCNP & VIM
    - prepared for lots of splitting & variance reduction
    - potential for lots of secondary particles

# Reproducibility & Parallel Calculation

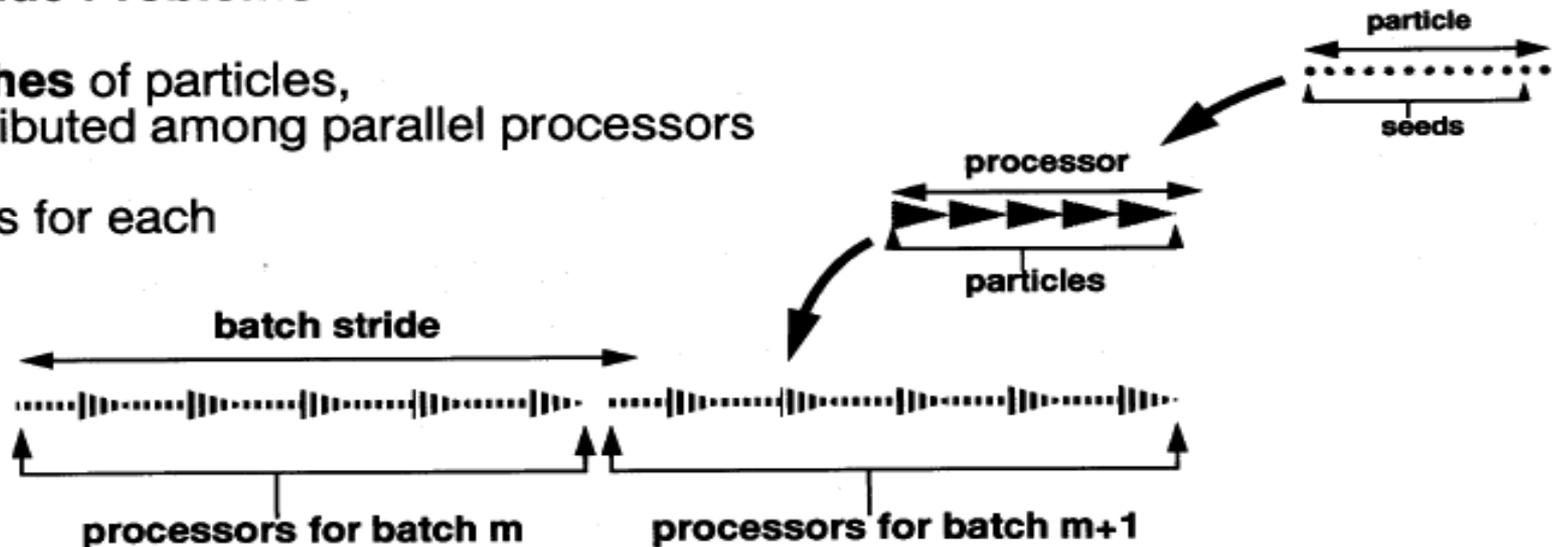
## Parallel processing

- take "super-stride" in random sequence for particles on each processor



## Eigenvalue Problems

- **batches** of particles, distributed among parallel processors
- seeds for each



# RNG Coding – Some Details

---

- **Real arithmetic**

- **IEEE Standard for Floating Point Arithmetic (IEE-754-2008)**
- **32-bit reals**
  - 24 bits of precision, ~7 decimal digits, max exponent ~38
  - **Never** use for general engineering/scientific calculations
- **64-bit reals**
  - 53 bits of precision, ~16 decimal digits, max exponent ~308
- **Arithmetic**
  - Least-significant bits discarded
  - For mixed ops, such as  $a*x+b$ , intermediate results may retain more bits

- **Integer arithmetic**

- **32-bit integers** – Fortran integer, C++ int
- **64-bit integers** – Fortran integer(8), C++ long or 'long long'
- C++ allows unsigned integers, Fortran does not
- If overflow in arithmetic, least significant bits are retained
- For  $(\text{long } a)*(\text{long } b)$ , rightmost 64-bits are kept
- Bits in an integer are conventionally numbered right-to-left

63 62 61 ... 3 2 1 0

# RNG Coding – Some Details

---

- **RNGs generate integer sequences**

- Integers converted to reals (fractions) for use in MC codes

- By convention, RNGs should not return exactly 0.0 or 1.0,  
 **$0.0 < \text{rang}() < 1.0$**

- It should be safe to do this:

**$\log(\text{rang}())$**

- Or this:

**$1. / \text{rang}()$**

- This should return an integer in  $[0, N-1]$ :

**$\text{floor}(N * \text{rang}())$**

- **Mixed LCGs include 0 in the integer sequence**

- Must return smallest positive real number if that occurs

- **Why use 63-bit RNGs, instead of 64-bit ?**

- **In Fortran, all integers are signed – there are no unsigned types**

- Largest positive integer is  $2^{63} - 1$ , **63 bits**

- Could of course use tricks to get around this limitation, but then portability to different compilers becomes a serious issue

- **Prefer to use an RNG that can be written in either Fortran or C++**

# RNG Coding – Some Details

---

- **Multiplying & adding 63-bit integers**

- Below, `RN_MASK` is a 64-bit integer, 0 followed by 63 1s: `0111...111`

- C++ using 'unsigned long'

```

RN_SEED = ( RN_MULT * RN_SEED + RN_ADD ) & RN_MASK ;

```

- Fortran, using 8-byte integers (signed)

```

RN_SEED = iand( RN_MULT * RN_SEED, RN_MASK )
RN_SEED = iand( RN_SEED + RN_ADD, RN_MASK )

```

Need to mask-off sign bit after each op

# RNG Coding – Some Details

---

- How do you convert a 63-bit integer to a real with 53-bit precision?

- Naive implementation

```
RN_MULT = pow( 2.0, -63 );  
rang = RN_SEED * RN_MULT ;
```

- Problem:

- RN\_SEED is in range  $[ 0, 2^{63}-1 ]$
- For the highest 512 integers in the range, the above approach results in `rang==1.0`. [ie, roundoff due to finite precision to exactly 1.0]

- Correct approach:

```
RN_MULT = pow( 2.0, -53 );  
i53 = RN_SEED >> 10           // shift right 10 bits, to get 53-bits  
if( ! i53 ) i53++;             // guard against 0  
rang = i53 * RN_MULT ;         // exact conversion to real
```

# RNG Coding – Some Details

- MCNP RNG (#2, 63-bit, simplified)

Note: bits are conventionally numbered right-to-left, bit-0 = rightmost, bit-63 = leftmost

```
unsigned long    RN_SEED,    I53;

// multiplier & adder
unsigned long    RN_MULT    = 9219741426499971445UL;
unsigned long    RN_ADD     = 1UL;
// mask to retain bits 0-62, 0 for bit 63
unsigned long    RN_MASK    = (1UL<<63) - 1UL;
// shift right 10 bits, retain most significant 53 bits
int              RN_SHIFT   = 10;
// multiplier to convert 53-bit int to double, 2.0**(-53)
double          RN_NORM     = 1.0 / (double) (1UL<<53);

// new 63-bit integer seed
RN_SEED = ( RN_MULT * RN_SEED + RN_ADD ) & RN_MASK;
// convert to double, 53-bit precision
I53      = RN_SEED >> RN_SHIFT;
if( ! I53 ) I53++; // guards against 0
return (double) ( I53 * RN_NORM );
```

# Skip-ahead for LCGs

---

- To skip ahead  $k$  steps in the RN sequence:

$$\begin{aligned} S_k &= g S_{k-1} + c \pmod{2^m} \\ &= g^k S_0 + c (g^k - 1) / (g - 1) \pmod{2^m} \end{aligned}$$

- Negative skip  $k$  equivalent to positive skip  $[\text{period}-k]$
- Can skip from any seed to any other
  - Initial seed  $\rightarrow$   $i^{\text{th}}$  seed for  $j^{\text{th}}$  particle on  $m^{\text{th}}$  processor in  $k^{\text{th}}$  generation
  - Particle  $i \rightarrow$  particle  $j$
  - Batch  $i \rightarrow$  batch  $j$
- Need a fast way to compute  $g^k \pmod{2^m}$  &  $c(g^k - 1) / (g - 1) \pmod{2^m}$  in  $O(m)$  steps, rather than  $O(k)$  steps

Reference: F.B. Brown, "Random Number Generation with Arbitrary Strides",  
*Trans. Am. Nucl. Soc.* (Dec 1994)

# Random Number Generators - Skip Ahead

To skip ahead  $k$  steps in the random sequence, [initial seed]  $\rightarrow$  [ $k^{\text{th}}$  seed]

$$\begin{aligned}
 S_k &= g \cdot S_{k-1} + c \pmod{2^m} \\
 &= g \cdot (g \cdot S_{k-2} + c) + c \pmod{2^m} \\
 &= g(\dots g(g(gS_0 + c) + c) \dots) + c \pmod{2^m} \\
 &= g^k \cdot S_0 + c \cdot (g^{k-1} + g^{k-2} + \dots + g + 1) \pmod{2^m} \\
 &= g^k \cdot S_0 + c \cdot (g^k - 1) / (g - 1) \pmod{2^m}
 \end{aligned}$$

- Periodic sequence:  
negative skip  $k_n$  equivalent to positive skip (**period -  $k_n$** )
- Can skip from any seed directly to any other:  
initial seed  $\rightarrow$   $i^{\text{th}}$  seed for  $j^{\text{th}}$  particle on  $m^{\text{th}}$  processor in  $n^{\text{th}}$  batch  
particle  $i \rightarrow$  particle  $j$   
batch  $i \rightarrow$  batch  $j$
- All arithmetic must be performed  $\pmod{2^m}$ , without truncation or roundoff

$$S_k = G(k) \cdot S_0 + C(k) \pmod{2^m}$$

# Random Number Generators - Skip Ahead

---

Define  $G(k) = g^k \bmod 2^m$

$m = 32$  or  $48$  (typical), based on the size of a computer word  
 $-2^m < k < +2^m$ , based on desired "stride"

Denote the  $j^{\text{th}}$  bit of  $k$  by  $k_{[j]}$ , so that

$$k = 2^{m-1} k_{[m-1]} + 2^{m-2} k_{[m-2]} + \dots + 2^1 k_{[1]} + 2^0 k_{[0]}$$

Substituting into  $G(k)$  yields

$$\begin{aligned} G(k) &= g^k \bmod 2^m = g^{\sum_{j=0}^{m-1} k_{[j]} 2^j} \bmod 2^m \\ &= \prod_{j=0}^{m-1} \left( g^{2^j} \right)^{k_{[j]}} \bmod 2^m \end{aligned}$$

Efficient algorithms for evaluating  $G(k)$  can be formulated using only  $m$  steps

# Random Number Generators - Skip Ahead

Enumerating a few terms of  $G(k)$  makes the algorithm obvious

$$G(k) = \left(g^1\right)^{k_{[0]}} \cdot \left(g^2\right)^{k_{[1]}} \cdot \left(g^4\right)^{k_{[2]}} \cdot \left(g^8\right)^{k_{[3]}} \cdots \left(g^{2^{m-1}}\right)^{k_{[m-1]}} \pmod{2^m}$$

Note that  $k_{[j]}=0$  or  $k_{[j]}=1$ , so that each term  $\left(g^n\right)^{k_{[j]}}$  evaluates to either 1 or  $g^n$

## Algorithm G:

```

G ← 1,    h ← g,    i ← k + 2m mod 2m
while  i > 0
    if  i = odd:    G ← G h mod 2m
    h ← h2 mod 2m
    i ← ⌊ i / 2 ⌋
  
```

## Remarks

- Algorithm G terminates after  $m$  steps, rather than  $k$  steps
- Negative strides are trivial, due to periodicity:  $G(-s) = G(2^m - s)$

# Random Number Generators - Skip Ahead

Define

$$C(k) = c \left( \frac{g^k - 1}{g - 1} \right) \bmod 2^m$$

$$= c \cdot \left( 1 + g + g^2 + g^3 + \dots + g^{k-1} \right) \bmod 2^m$$

The series for  $C(k)$  can be evaluated recursively, similar to  $G(k)$ , in  $m$  steps:

## Algorithm C:

```

C ← 0,  f ← c,  h ← g,  i ← k + 2m mod 2m
while  i > 0
    if  i = odd:    C ← C h + f mod 2m
    f ← f (h + 1) mod 2m
    h ← h2 mod 2m
    i ← ⌊ i / 2 ⌋
  
```

- Since most of the common random number generators use  $c = 0$ , Algorithm C is generally not required.
- Algorithm C can be included with Algorithm A, at very little extra cost

# Random Number Generators - Skip Ahead - Example

R. N. Generator for 32-bit machines

(*sparc2, rs6000, indigo, .....*)

$$s \leftarrow 69069 \cdot s + 1 \pmod{2^{32}}$$

Random Number Generator →

```
static unsigned long    seed_c=1;
static double          norm=(1./4294967296.);

double cranf_( void ) {
    unsigned long    g=69069, c=1;
    seed_c = g * seed_c + c;
    return  ((double) seed_c * norm );
}
```

Routine for Arbitrary Skips →

```
void cranfjump_(    unsigned long    *seed,
                   double          *jump,
                   unsigned long    *newseed ) {
    unsigned long    j, gen=1, inc=0, g=69069, c=1;
    if( *jump < 0 )  j = *jump + 4294967296.;
    else             j = *jump;
    for( ; j; j>>=1 ) {
        if( j&1 ) {
            inc = inc*g + c;
            gen = gen*g;
        }
        c *= g+1;
        g *= g;
    }
    *newseed = gen * (*seed) + inc;
}
```

Compute:

$$gen = g^k$$

$$inc = c(g^k - 1)/(g - 1)$$

# Random Number Generators - Skip Ahead - Example

Fortran, 48-bit generator:  $g=5^{19}$ ,  $c=0$ ,  $m=48$  (VIM & MCNP)  
 C, 32-bit generator:  $g=69069$ ,  $c=1$ ,  $m=32$  (from Marsaglia)

		<u>Sparc2</u>	<u>rs6000/350</u>
<b>C, 32-bit</b>			
random number		1.0 $\mu$ s	.7 $\mu$ s
skip forward,	average for $+1\dots 10^5$	7.4 $\mu$ s	10 $\mu$ s
skip backward,	average for $-1\dots -10^5$	4.0 $\mu$ s	20 $\mu$ s
<b>Fortran, 48-bit</b>			
random number		3.6 $\mu$ s	2.3 $\mu$ s
skip forward,	+152,917	163 $\mu$ s	78 $\mu$ s
skip backward,	-152,917	458 $\mu$ s	215 $\mu$ s
skip forward,	average for $+1\dots 10^5$	160 $\mu$ s	75 $\mu$ s
skip backward,	average for $-1\dots -10^5$	695 $\mu$ s	232 $\mu$ s
skip forward,	+1,152,917	189 $\mu$ s	90 $\mu$ s
skip forward,	+1,152,917, <b>brute force</b>	4.1 <b>sec</b>	2.6 <b>sec</b>
skip backward,	-1,152,917	456 $\mu$ s	210 $\mu$ s
skip backward,	-1,152,917, <b>brute force</b>	8 <b>year</b>	5 <b>year</b>

# Random Number Generators - Skip Ahead

---

- Algorithms for direct skip-ahead in the random sequence are simple, fast, convenient, ....., for modern Monte Carlo codes
- Arbitrary positive or negative strides can be taken, without precomputing or hardwiring specific constants
- Direct skip-ahead simplifies the initialization of random numbers for each particle, especially for parallel processing
- Algorithms described are currently used in:

parallel VIM	— ANL	— Sun, rs6000, SP1, .....
RACER	— KAPL	— Cray, Meiko CS1 & CS2, Sun, SGI, .....
KENO-Va	— CSN (Spain)	— Convex-C3440
MCNP5	--- LANL	--- all machines

**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Random Number Generator Testing**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Random Number Generation & Testing

---

- **Knuth statistical tests**
- **Marsaglia's DIEHARD test suite**
- **Spectral test**
- **Performance test**
- **Results**

F.B. Brown & Y. Nagaya, "The MCNP5 Random Number Generator",  
*Trans. Am. Nucl. Soc.* [also, LA-UR-02-3782] (November, 2002)

Y. Nagaya & F.B. Brown, "Testing MCNP Random Number Generators",  
LANL report on testing MCNP5 RN generators,  
work performed in 2002 for original MCNP5 version, LA-UR-11-04858 (2011)

# MCNP5 RNG: History

---

- **MCNP & related precursor codes**

- 40+ years of intense use
- Many different computers & compilers
- Modern versions are parallel: MPI + threads
- History based: Consecutive RNs used for primary particle, then for each of it's secondaries in turn, etc.
- RN generator is small fraction of total computing time (~ 5%)

- **Traditional MCNP RN Algorithm**

- Linear congruential, multiplicative

$$S_{n+1} = g S_n \text{ mod } 2^{48}, \quad g = 5^{19}$$

- 48-bit integer arithmetic, carried out in 24-bit pieces
- Stride for new histories: 152,917
- Skip-ahead: crude, brute-force
- Period / stride =  $460 \times 10^6$  histories
- Similar RN generators in RACER, RCP, MORSE, KENO, VIM

# MCNP5 RNG: Requirements

---

- **Algorithm**

- Robust, well-proven
- Long period:  $> 10^9$  particles x stride 152,917 =  $10^{14}$  RNs
- $>10^9$  parallel streams
- High-precision is not needed, low-order bits not important
- Must have fast skip-ahead procedure
- Reasonable theoretical basis, no correlation within or between histories

- **Coding**

- Robust !!!! Must never fail.
- Rapid initialization for each history
- Minimal amount of state information
- Fast, but portable – must be exactly reproducible on any computer/compiler



# MCNP5 RNG: Coding

---

- **RN Generation in MCNP-5**
  - RN module, entirely replaces all previous coding for RN generation
  - Fortran-90, using INTEGER(I8) internally, where I8=selected\_int\_kind(18)
  - All parameters, variables, & RN generator state are PRIVATE, accessible only via “accessor” routines
  - Includes “new” skip-ahead algorithm for fast initialization of histories, greatly simplifies RN generation for parallel calculations
  - Portable, standard, thread-safe
  - Built-in unit test, compile check, and run-time test
  - Developed on PC, tested on SGI, IBM, Sun, Compaq, Mac, alpha

## Extended generators : 63-bit LCGs

- Selection of multiplier, increment and modulus

$$S_{n+1} = 5^{19} S_n + 0 \pmod{2^{48}} \quad (\text{MCNP4})$$

  
 $5^{23}, 5^{25}$

  
 $1$

  
 $2^{63}$

- Multiplicative LCG( $g, 0, 2^b$ )

$$g \equiv \pm 3 \pmod{8}, S_0 = \text{odd} \quad \Rightarrow \quad \text{Period} : 2^{b-2}$$

- Mixed LCG( $g, c, 2^b$ )

$$g \equiv 1 \pmod{4}, c = \text{odd} \quad \Rightarrow \quad \text{Period} : 2^b$$

- MCNP5 - Extension of multiplier

- $5^{19}$  = 45-bit integer in the binary representation
- $5^{19}$  seems to be slightly small in 63-bit environment.
- Odd powers of 5 satisfy both conditions above.
- Try these:  $(5^{19}, 0, 2^{63}), (5^{23}, 0, 2^{63}), (5^{25}, 0, 2^{63}),$   
 $(5^{19}, 1, 2^{63}), (5^{23}, 1, 2^{63}), (5^{25}, 1, 2^{63})$

# L'Ecuyer's 63-bit LCGs

---

- **L'Ecuyer suggested 63-bit LCGs with good lattice structures.**  
**Math. Comp., 68, 249-260 (1999)**
  - **Good multipliers were chosen based on the spectral test.**
  - **Multiplicative LCGs**
    - LCG(3512401965023503517, 0,  $2^{63}$ )
    - LCG(2444805353187672469, 0,  $2^{63}$ )
    - LCG(1987591058829310733, 0,  $2^{63}$ )
  - **Mixed LCGs**
    - LCG(9219741426499971445, 1,  $2^{63}$ )
    - LCG(2806196910506780709, 1,  $2^{63}$ )
    - LCG(3249286849523012805, 1,  $2^{63}$ )

# Tests for RNGs

---

- **13 different LCGs were tested:**
  - Traditional MCNP RNG, ( $5^{19}$ , 0,  $2^{48}$ )
  - 6 - Extended 63-bit LCGs
  - 6 - L'Ecuyer's 63-bit LCGs
- **Theoretical tests :**
  - Analyze the RNG algorithm of based on number theory and the theory of statistics.
  - Theoretical tests depend on the type of RNG. (LCG, Shift register, Lagged Fibonacci, etc.)
  - For LCGs, the **Spectral test** is used
- **Empirical tests :**
  - Analyze the uniformity, patterns, etc. of RNs generated by RNGs .
  - **Standard tests** - reviewed by D. Knuth, SPRNG test routines
  - **DIEHARD tests** - Bit level tests by G. Marsaglia, more stringent
  - **Physical tests** - RNGs are used in a practical application. The exact solutions for the tests are known. (not performed in this work)

# Spectral test

---

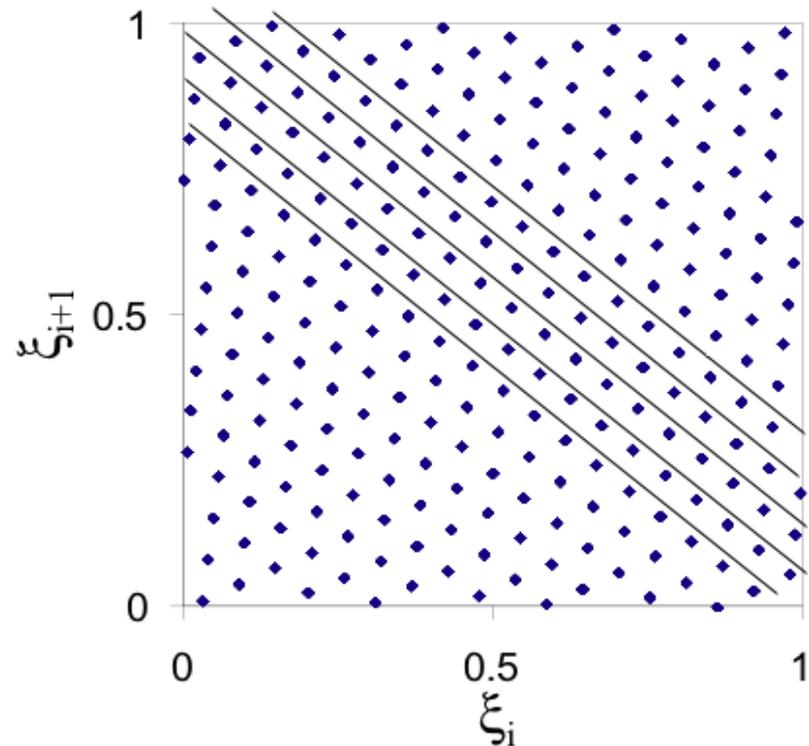
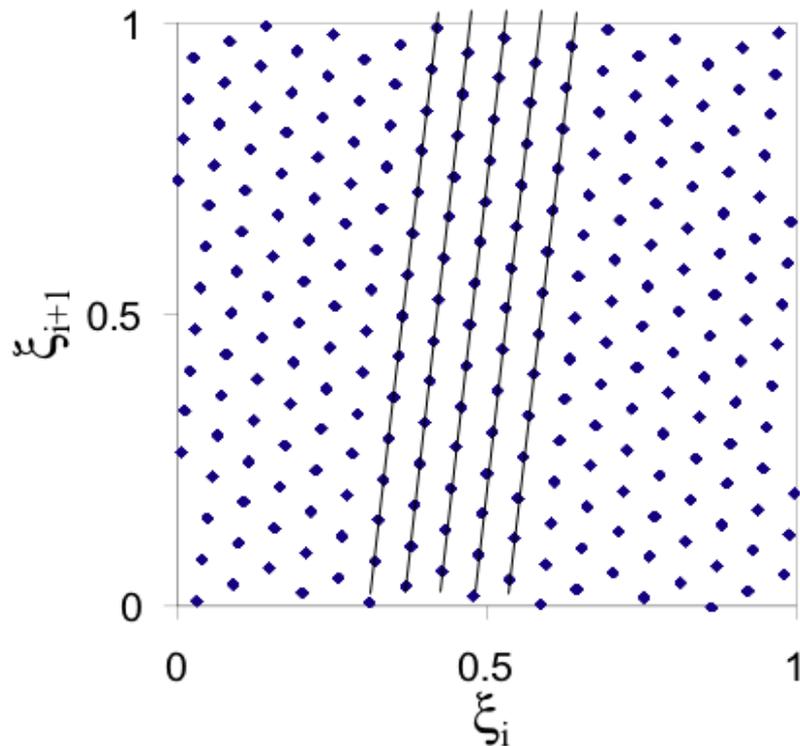
- LCGs have regular patterns (lattice structures) when overlapping  $t$ -tuples of a random number sequence are plotted in a hypercube. (Marsaglia, 1968).
- all the  $t$ -tuples are covered with families of parallel  $(t-1)$ -dimensional hyperplanes.
- The spectral test determines the maximum distance between adjacent parallel hyperplanes.

# Illustration of the spectral test

- Example:  $S_{n+1} = 137 S_n + 187 \pmod{256}$

$$0.26562, 0.12109, 0.32031, 0.61328, 0.75000, \dots$$

(The sequence is grouped into four pairs: (0.26562, 0.12109), (0.32031, 0.61328), (0.75000, ...), and an implied pair for the first two terms.)



# Measures for Spectral Test Criterion & Ranking

- $\mu$  value proposed by Knuth
  - Represent the effectiveness of a multiplier.

## Knuth's criterion

$\mu_t(m,g)$ for $2 \leq t \leq 6$	Result
$\mu_t(m,g) > 1$	Pass with flying colors
$0.1 \leq \mu_t(m,g) \leq 1$	Pass
$\mu_t(m,g) \leq 0.1$	Fail

- **S value**
  - Normalized maximum distance.

$$S_t = \frac{d_t^*(m)}{d_t(m, g)}$$

$d_t(m, g)$ : Maximum distance between adjacent parallel hyperplanes.  
 $d_t(m)$ : Lower bound on  $d_t(m, g)$ .

- The closer to 1 the S value is, the better the RNG is.

# Spectral test for extended LCGs

Dimension(t)	2	3	4	5	6	7	8
<b>LCG(5<sup>19</sup>,0,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	1.7321	2.1068	2.7781	1.4379	0.0825	2.0043	5.9276
$S_t(m,g)$	0.6910	0.7085	0.7284	0.6266	0.3888	0.6573	0.7414
<b>LCG(5<sup>23</sup>,0,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	0.0028	1.9145	2.4655	5.4858	0.3327	0.2895	6.6286
$S_t(m,g)$	0.0280	0.6863	0.7070	0.8190	0.4906	0.4986	0.7518
<b>LCG(5<sup>25</sup>,0,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	0.3206	1.8083	0.0450	3.0128	0.3270	3.1053	0.4400
$S_t(m,g)$	0.2973	0.6733	0.2598	0.7265	0.4892	0.6998	0.5356
<b>LCG(5<sup>19</sup>,1,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	1.7321	2.9253	2.4193	0.3595	0.0206	0.5011	1.6439
$S_t(m,g)$	0.6910	0.7904	0.7036	0.4749	0.3086	0.5392	0.6316
<b>LCG(5<sup>23</sup>,1,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	0.0007	2.8511	2.5256	3.1271	4.5931	1.8131	4.2919
$S_t(m,g)$	0.0140	0.7837	0.7112	0.7319	0.7598	0.6480	0.7121
<b>LCG(5<sup>25</sup>,1,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	0.0801	3.4624	1.3077	1.0853	1.4452	0.7763	1.3524
$S_t(m,g)$	0.1486	0.8361	0.6033	0.5923	0.6266	0.5740	0.6163

# Spectral test for L'Ecuyer's 63-bit LCGs

Dimension(t)	2	3	4	5	6	7	8
<b>LCG(3512401965023503517,0,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	2.9062	2.9016	3.1105	4.0325	5.3992	6.7498	7.2874
$S_t(m,g)$	0.8951	0.7883	0.7493	0.7701	0.7806	0.7818	0.7608
<b>LCG(2444805353187672469,0,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	2.2588	2.4430	6.4021	2.9364	3.0414	5.4274	4.6180
$S_t(m,g)$	0.7891	0.7443	0.8974	0.7228	0.7094	0.7579	0.7186
<b>LCG(1987591058829310733,0,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	2.4898	3.4724	1.7071	2.5687	2.1243	2.0222	4.1014
$S_t(m,g)$	0.8285	0.8369	0.6449	0.7037	0.6682	0.6582	0.7080
<b>LCG(9219741426499971445,1,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	2.8509	2.8046	3.5726	3.8380	3.8295	6.4241	6.8114
$S_t(m,g)$	0.8865	0.7794	0.7757	0.7625	0.7371	0.7763	0.7544
<b>LCG(2806196910506780709,1,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	1.9599	4.0204	4.4591	3.1152	3.0728	3.0111	3.7947
$S_t(m,g)$	0.7350	0.8788	0.8199	0.7314	0.7106	0.6967	0.7012
<b>LCG(3249286849523012805,1,2<sup>63</sup>)</b>							
$\mu_t(m,g)$	2.4594	2.4281	3.7081	2.8333	3.7633	3.0844	1.9471
$S_t(m,g)$	0.8234	0.7428	0.7829	0.7176	0.7350	0.6991	0.6451

# Results of spectral test

- Results for the traditional MCNP RNG**

Dimension(t)	2	3	4	5	6	7	8
$\mu_t(m,g)$	3.0233	0.1970	1.8870	0.9483	1.8597	0.8802	1.2931
$S_t(m,g)$	0.9129	0.3216	0.6613	0.5765	0.6535	0.5844	0.6129

- All extended 63-bit LCGs fail with Knuth's criterion.**
- All L'Ecuyer's 63-bit LCGs pass with flying colors.**
- Comparison of minimum S values**

RNG	Minimum $S_t(m,g)$
<b>LCG(<math>5^{19},0,2^{48}</math>)</b>	<b>0.3216</b>
<b>LCG(<math>3512401965023503517,0,2^{63}</math>)</b>	0.7493
<b>LCG(<math>2444805353187672469,0,2^{63}</math>)</b>	0.7094
<b>LCG(<math>1987591058829310733,0,2^{63}</math>)</b>	0.6449
<b>LCG(<math>9219741426499971445,1,2^{63}</math>)</b>	0.7371
<b>LCG(<math>2806196910506780709,1,2^{63}</math>)</b>	0.6967
<b>LCG(<math>3249286849523012805,1,2^{63}</math>)</b>	0.6451

# Standard test suite in SPRNG

---

- **SPRNG (Scalable Parallel Random Number Generators )**
  - Test programs are available. <http://sprng.cs.fsu.edu>
- **Standard test suite (Knuth)**
  - Equidistribution
  - Serial
  - Gap
  - Poker
  - Coupon collector's
  - Permutation
  - Runs-up
  - Maximum-of-t
  - Collision tests
- **Choice of test parameters**
  - L'Ecuyer's test suite : Comm. ACM 31 p.742 (1988)
  - Vattulainen's test suite : Comp. Phys. Comm. 86 p.209 (1995)
  - Mascagni's test suite : Submitted to Parallel Computing

# Equidistribution test

- Check whether RNs are uniformly generated in  $[0, 1)$ .
- Generate random integers in  $[0, d-1]$ .
- Each integer must have the equal probability  $1/d$ .

0.10574, 0.66509, 0.46622, 0.93925, 0.26551, 0.11361, ...

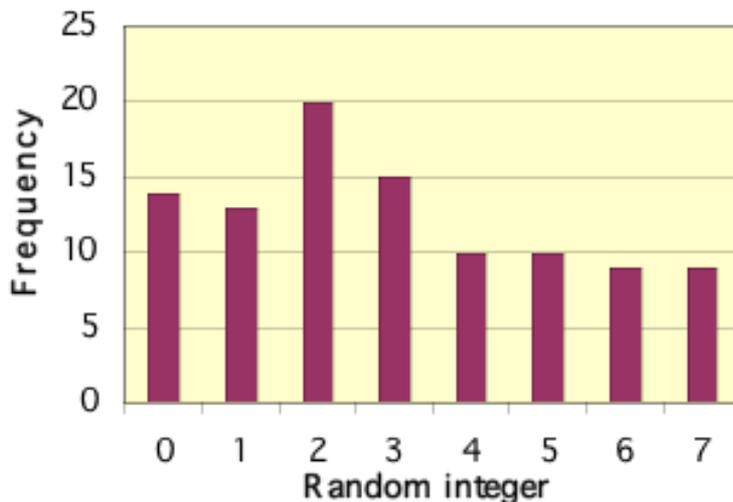


$d * x_i$

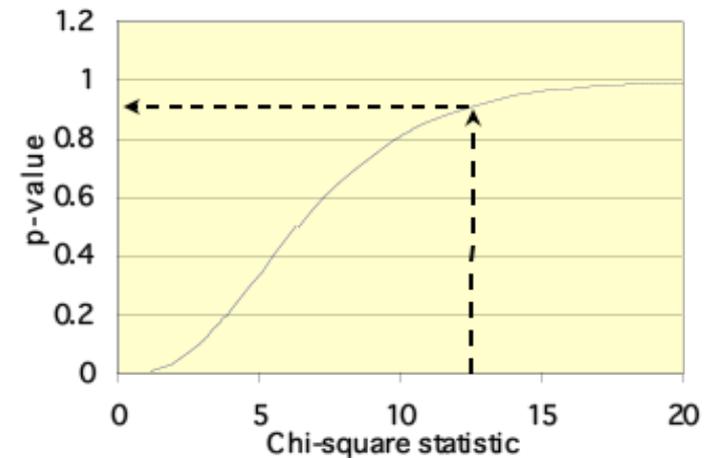
0, 5, 3, 7, 2, 0, 2, 3, 1, 4, ...



Count frequencies of  $0 \sim d-1$ .



Cumulative chi-square distribution

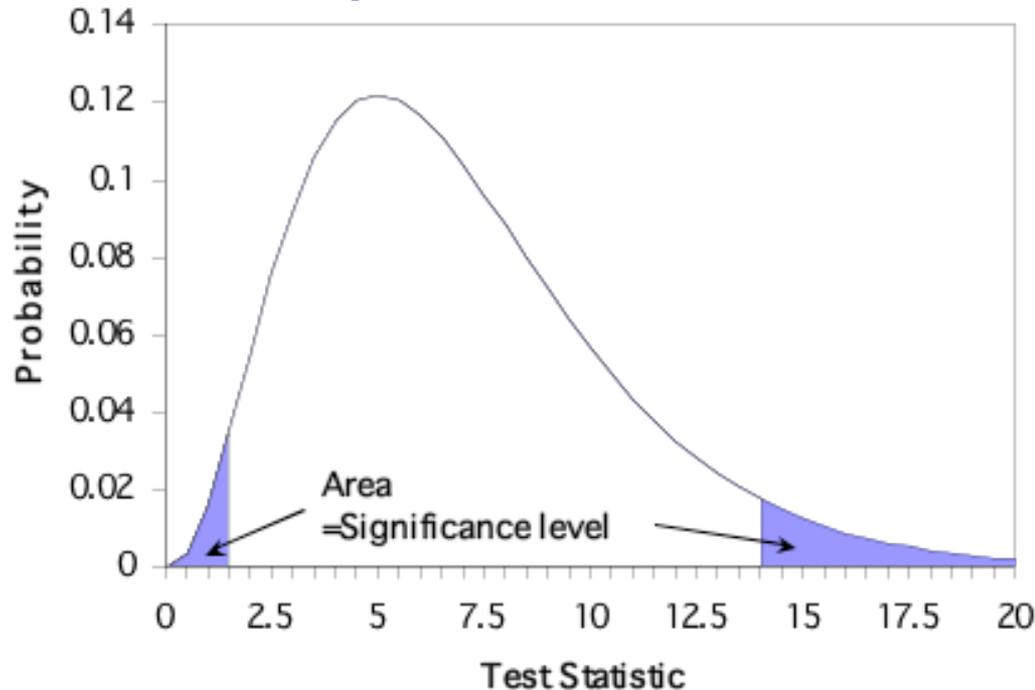


$$V = \sum_{s=1}^k \frac{(Y_s - np_s)^2}{np_s}$$

## Criterion of “Pass or Failure”

---

- All empirical tests score a statistic.
- A goodness-of-fit test is performed on the test statistic and yield a p-value. (Chi-square or Kolmogorov-Smirnov test)
- If the p-value is close to 0 or 1, a RNG is suspected to fail.
- Significance level : 0.01(1%)
- Repeat each test 3 times.
- If all 3 p-values are suspicious, then the RNG fails.



# DIEHARD test suite

---

- **DIEHARD test**
  - A battery of tests proposed by G. Marsaglia.
  - Test all bits of random integers, not only the most significant bits.
  - More stringent than standard Knuth tests.
  - Default test parameters were used in this work.
  - Test programs are available. <http://stat.fsu.edu/~geo/diehard.html>
- **Included tests:**
  - Birthday spacings
  - Overlapping 5-permutation
  - Binary rank
  - Bitstream
  - Overlapping-pairs-sparse-occupancy (OPSO)
  - Overlapping-quadruples-sparse-occupancy (OQSO)
  - DNA
  - Count-the-1's test on a stream of bytes
  - Count-the-1's test for specific bytes
  - Parking lot
  - Minimum distance
  - 3-D spheres
  - Squeeze
  - Overlapping sums
  - Runs
  - Craps

# Overlapping-pairs-sparse-occupancy test (1)

---

- OPSO = Overlapping-Pairs-Sparse-Occupancy test
- Preparation of 32-bit integers

0.10574, 0.66509, 0.46622, 0.93925, 0.26551, 0.11361, ...

↓  $2^{32} * x_i$

454158374, 2856527213, 2002411287, 4034027575, ...

↓ Binary representation

11011000100011110100000100110,  
10101010010000110010010101101101, ...

- Letter : a designated string of consecutive 10 bits

1101100010001111010**0000100110**,  
1010101001000011001001**0101101101**, ...

Letter :  $2^{10} = 1024$  patterns (letters)

## Overlapping-pairs-sparse-occupancy test (2)

- 2-letter words are formed from an alphabet of 1024 letters.

0000100110, 0101101101, 1100010111, 0000110111, ...



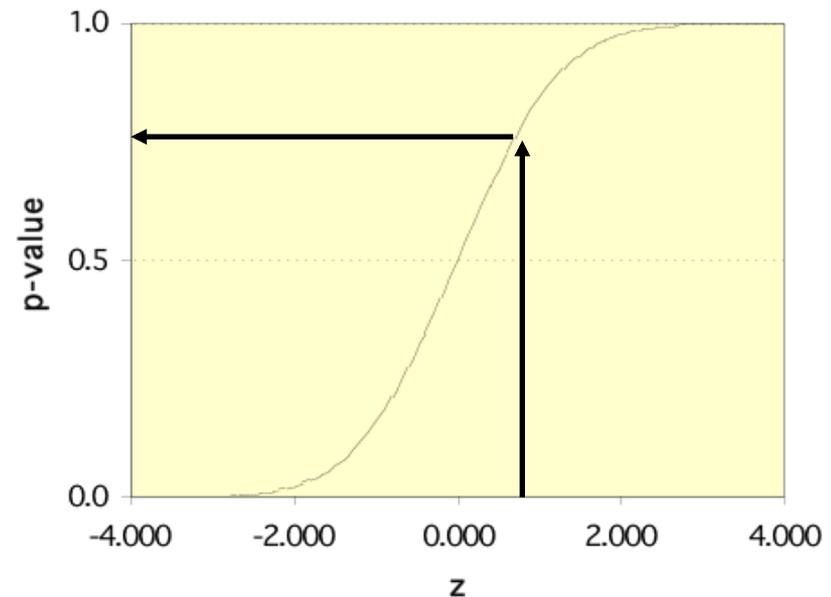
Decimal representation

38, 365, 791, 55, ...

2-letter word 2-letter word

- Count the number of missing words ( $=j$ ).
- The number of missing words should be very closely normally distributed with mean 141,909, standard deviation 290.

Cumulative normal distribution



$$z = \frac{j}{290} \frac{141909}{290}$$

# Overlapping-quadruples-sparse-occupancy test

---

- **OQSO = Overlapping-Quadruples-Sparse-Occupancy test**
- **Similar to the OPSO test.**
- **Letter : a designated string of consecutive 5 bits**

110110001000111101000001**00110**,  
 101010100100001100100101011**01101**, ...

Letter :  $2^5 = 32$  letters

- **4-letter words are formed from an alphabet of 32 letters.**

**00110, 01101, 10111, 10111, ...**

4-letter word

- **The number of missing words should be very closely normally distributed with mean 141909, standard deviation 295.**

# DNA test

---

- Similar to the OPSO and OQSO tests.
- Letter : a designated string of consecutive 2 bits

110110001000111101000001001**10**,  
 101010100100001100100101011011**01**, ...

  
 Letter :  $2^2 = 4$  letters

- 10-letter words are formed from an alphabet of 4 letters.

**10, 1, 11, 11, 11, 1, 10, 0, 11, 10**, ...

  
 10-letter word

- The number of missing words should be very closely normally distributed with mean 141909, standard deviation 399.

# DIEHARD Test Suite

---

- **Criterion for DIEHARD test**

- If the p-value is close to 0 or 1, a RNG is suspected to fail.
- Significance level : 0.01(1%)
- A RNG fails the test if we get six or more p-values less than 0.01 or more than 0.99.

- **Results for standard & DIEHARD tests**

- All 13 RNGs **pass** all standard tests with L'Ecuyer's, Vattulainen's and Mascagni's test parameters.
- Extended and L'Ecuyer's 63-bit LCGs **pass** all the DIEHARD tests.
- The traditional MCNP RNG **fails** the OPSO, OQSO and DNA tests in the DIEHARD test suite.

# Result of OPSO test for traditional MCNP RNG

Tested bits	p-value	Tested bits	p-value
bits 23 to 32	<b>0.0000</b>	bits 11 to 20	0.7457
bits 22 to 31	<b>0.0000</b>	bits 10 to 19	0.0598
bits 21 to 30	<b>0.0000</b>	bits 9 to 18	0.1122
bits 20 to 29	<b>0.0000</b>	bits 8 to 17	0.4597
bits 19 to 28	<b>0.0001</b>	bits 7 to 16	<b>0.0011</b>
bits 18 to 27	0.6639	bits 6 to 15	0.6319
bits 17 to 26	0.0445	bits 5 to 14	0.7490
bits 16 to 25	0.0125	bits 4 to 13	0.2914
bits 15 to 24	0.7683	bits 3 to 12	0.1792
bits 14 to 23	0.9712	bits 2 to 11	0.3253
bits 13 to 22	0.1077	bits 1 to 10	0.7277
bits 12 to 21	0.0717		

# Result of QQSO test for traditional MCNP RNG

Tested bits	p-value	Tested bits	p-value
bits 28 to 32	<b>1.0000</b>	bits 14 to 18	0.6487
bits 27 to 31	<b>1.0000</b>	bits 13 to 17	0.5575
bits 26 to 30	<b>1.0000</b>	bits 12 to 16	0.1634
bits 25 to 29	<b>1.0000</b>	bits 11 to 15	0.6600
bits 24 to 28	<b>1.0000</b>	bits 10 to 14	0.2096
bits 23 to 27	<b>1.0000</b>	bits 9 to 13	0.3759
bits 22 to 26	<b>0.0000</b>	bits 8 to 12	0.9191
bits 21 to 25	<b>0.0000</b>	bits 7 to 11	0.8554
bits 20 to 24	<b>0.0000</b>	bits 6 to 10	0.5535
bits 19 to 23	0.1906	bits 5 to 9	0.4955
bits 18 to 22	<b>0.0011</b>	bits 4 to 8	0.0868
bits 17 to 21	0.3823	bits 3 to 7	0.1943
bits 16 to 20	0.8394	bits 2 to 6	0.8554
bits 15 to 19	0.2518	bits 1 to 5	0.7421

# Result of DNA test for traditional MCNP RNG

Tested bits	p-value	Tested bits	p-value	Tested bits	p-value
bits 31 to 32	<b>1.0000</b>	bits 20 to 21	0.4937	bits 9 to 10	0.4550
bits 30 to 31	<b>1.0000</b>	bits 19 to 20	0.0613	bits 8 to 9	0.4737
bits 29 to 30	<b>1.0000</b>	bits 18 to 19	0.2383	bits 7 to 8	0.7834
bits 28 to 29	<b>1.0000</b>	bits 17 to 18	0.4831	bits 6 to 7	0.4063
bits 27 to 28	<b>1.0000</b>	bits 16 to 17	0.0925	bits 5 to 6	0.8959
bits 26 to 27	0.1777	bits 15 to 16	0.0197	bits 4 to 5	0.3438
bits 25 to 26	<b>0.0000</b>	bits 14 to 15	0.7377	bits 3 to 4	0.3972
bits 24 to 25	<b>0.0000</b>	bits 13 to 14	0.7171	bits 2 to 3	0.8986
bits 23 to 24	<b>0.0000</b>	bits 12 to 13	0.0309	bits 1 to 2	0.5407
bits 22 to 23	<b>0.0000</b>	bits 11 to 12	0.2803		
bits 21 to 22	<b>0.0000</b>	bits 10 to 11	0.8440		

# Comments on results for OPSO, OQSO, DNA

---

- **Less significant (lower) bits of RNs fail the tests.**
- **These failures in less significant bits are caused by the shorter period than the significant bits.**

## **Drawback of LCGs with power-of-two modulus**

**The  $(r+1)$ -th most significant bit has period length at most  $2^{-r}$  times that of the most significant bit.**

- **However, these failures do not have a significant impact in the practical use.**

# Performance test

---

- Test program

```
integer(8) :: i
integer(8), parameter :: NumGeneratedRNs = 1000000000
!real(8)      :: rang ! For MCNP4
real(8)      :: RN_initial, RN_last
real(8)      :: dummy
. . . . .

!call random ! For MCNP4
call RN_init_problem( new_standard_gen = 1 )

RN_initial = rang()

do i = 2, NumGeneratedRNs-1
  dummy = rang()
end do

RN_last = rang()
. . . . .
```

# Results of performance test

---

- Comparison between MCNP4 and MCNP5
- Generate 1 billion RNs.

	<b>MCNP4</b>	<b>MCNP5</b>	<b>MCNP4/MCNP5</b>
<b>CPU (sec)</b> No optimization (/optimization:0)	<b>290.0</b>	<b>97.1</b>	<b>3.0</b>
<b>CPU (sec)</b> Local optimization (/optimization:1)	<b>191.7</b>	<b>77.2</b>	<b>2.5</b>
<b>CPU (sec)</b> Full optimization (/optimization:4)	<b>188.4</b>	<b>78.1</b>	<b>2.4</b>

**Platform : Windows 2000, Intel Pentium III 1GHz**

**Compiler : Compaq Visual Fortran Ver.6.6**

# Summary

---

- **The traditional MCNP RNG fails the OPSO, OQSO and DNA tests in the DIEHARD test suite.**
- **The 63-bit LCGs extended from the MCNP RNG fail the spectral test.**
- **L'Ecuyer's 63-bit LCGs pass all the tests and their multipliers are excellent judging from the spectral test.**
- **These 63-bit LCGs are implemented in the RNG package for MCNP5**
- **The MCNP5 RNG is ~2.5 times faster than the MCNP4 RNG.**

# Random Sampling -- References

---

**Every Monte Carlo code developer who works with random sampling should own & read these references:**

- D. E. Knuth, The Art of Computer Programming, Vol. 2: Semi-numerical Algorithms, 3<sup>rd</sup> Edition, Addison-Wesley, Reading, MA (1998).
- L. Devroye, Non-Uniform Random Variate Generation, Springer-Verlag, NY (1986).
- J. von Neumann, "Various Techniques Used in Conjunction with Random Digits," *J. Res. Nat. Bur. Stand. Appl. Math Series* 3, 36-38 (1951).
- C. J. Everett and E. D. Cashwell, "A Third Monte Carlo Sampler," LA9721-MS, Los Alamos National Laboratory, Los Alamos, NM (1983).
- H. Kahn, "Applications of Monte Carlo," AECU-3259, Rand Corporation, Santa Monica, CA (1954).
- F.B. Brown, "Random Number Generation with Arbitrary Strides", *Trans. Am. Nucl. Soc.* (Dec 1994)
- F.B. Brown & Y. Nagaya, "The MCNP5 Random Number Generator", *Trans. Am. Nucl. Soc.* [also, LA-UR-02-3782] (November, 2002)
- Y. Nagaya & F.B. Brown, "Testing MCNP Random Number Generators", LANL report on testing MCNP5 RN generators, work performed in 2002 for original MCNP5 version, LA-UR-11-04858 (2011)







**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Random Sampling – Beyond the Basics**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Outline

---

- **Introduction**
- **Random Sampling – Basics**
- **Linear Transformations & Scaling**
- **Composition Methods**
- **Rejection Methods**
- **Miscellaneous**

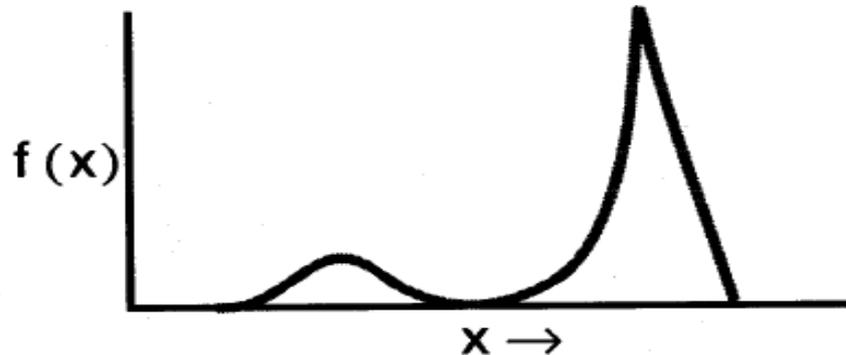
# Introduction

---

The key to Monte Carlo methods is the notion of *random sampling*.

- The problem can be stated this way:

Given a probability density,  $f(x)$ , produce a sequence of  $\hat{X}$ 's.  
The  $\hat{X}$ 's should be distributed in the same manner as  $f(x)$ .



- Random sampling distinguishes Monte Carlo from other methods
- When Monte Carlo is used to solve the Boltzmann transport equation:
  - Random sampling models the outcome of physical events (e.g., neutron collisions, fission process, sources, .....

# Monte Carlo & Random Sampling

---

## Categories of random sampling

- **Random number generator** → uniform PDF on (0,1)
- Sampling from **analytic PDFs** → normal, exponential, Maxwellian, ...
- Sampling from **tabulated PDFs** → angular PDFs, spectrum, ...

## For Monte Carlo codes...

- **Random numbers,  $\xi$ , are produced by the RN generator on (0,1)**
- **Non-uniform random variates are produced from the  $\xi$ 's by:**
  - Direct inversion
  - Rejection methods
  - Transformations
  - Composition (mixtures)
  - Sums, products, ratios, ...
  - Table lookup + interpolation
  - Lots (!) of other tricks
- **Typically < 5 - 10% of total CPU time**

---

# Random Sampling - Basics

**"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."**

**John Von Neuman, 1951**

# Introduction

---

## Probability ?

What are the odds of .....

- Being audited by the IRS this year 100 to 1
- Losing your luggage on a U.S. flight 176 to 1
- Being dealt 4 aces on an opening poker hand 4,164 to 1
- Being struck by lightning in your lifetime 9,100 to 1
- Being hit by a baseball at a major league game 300,000 to 1
- Drowning in your bathtub this year 685,000 to 1
- Winning the Powerball jackpot with 1 ticket 292,201,338 to 1

Yet we all still keep buying Powerball tickets, but don't worry too much about lightning...

# Simple Random Sampling (1)

- **Suppose we have 2 items, A and B**

- $P_A$  = probability of randomly picking item A = 25% = 0.25

- $P_B$  = probability of randomly picking item B = 75% = 0.75

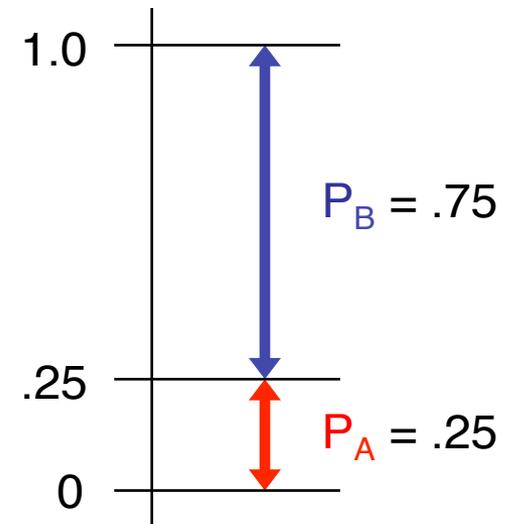
- $P_A + P_B = 1.0$

- **Random sampling - pick A or B**

Generate a random number  $\mathcal{R}$   
in the range (0,1)

If  $\mathcal{R} < .25 \rightarrow$       **select A**

Otherwise  $\rightarrow$       **select B**



Cumulative  
Probabilities

## Simple Random Sampling (2)

- **Suppose we have 3 items, A, B, and C**

- $P_A$  = probability of randomly picking item A = 25% = 0.25

- $P_B$  = probability of randomly picking item B = 50% = 0.50

- $P_C$  = probability of randomly picking item C = 25% = 0.25

- $P_A + P_B + P_C = 1.0$

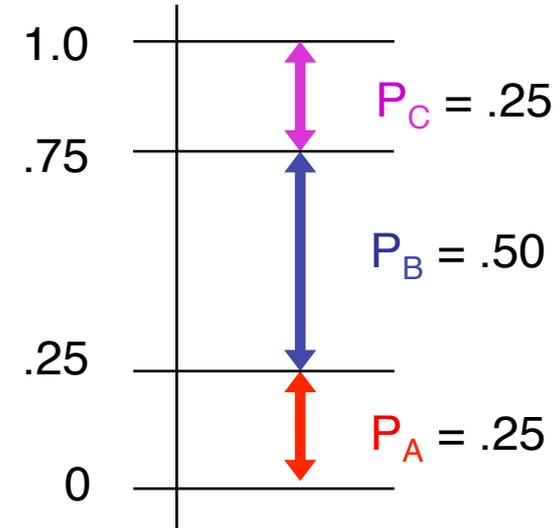
- **Random sampling - pick A or B or C**

Generate a random number  $\mathcal{R}$   
in the range (0,1)

If  $\mathcal{R} < .25$  → select A

Else If  $.25 < \mathcal{R} < .75$  → select B

Otherwise → select C

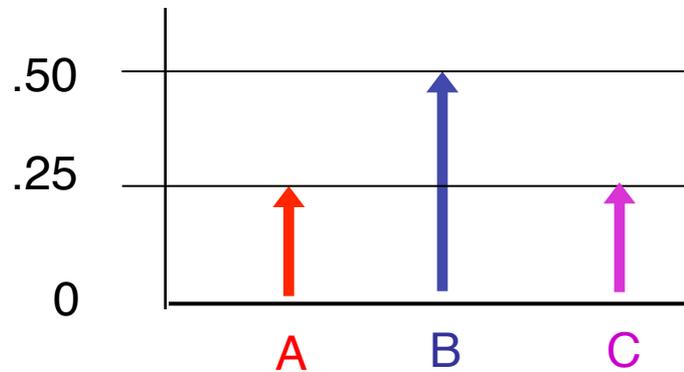


Cumulative  
Probabilities

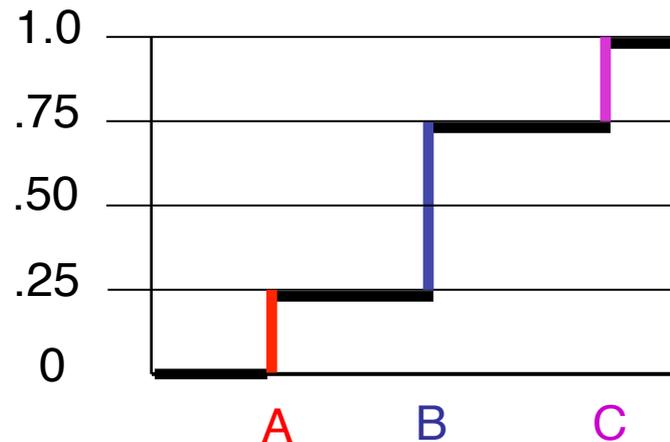
# Simple Random Sampling (3)

- **Random sampling - pick A or B or C**

- $P_A$  = probability of randomly picking item A = 25% = 0.25
- $P_B$  = probability of randomly picking item B = 50% = 0.50
- $P_C$  = probability of randomly picking item C = 25% = 0.25
- $P_A + P_B + P_C = 1.0$



Discrete  
Probabilities



Cumulative  
Probabilities

Generate a  
random number  $\mathcal{R}$   
in the range (0,1),  
Pick A, B, or C

# Probability Density Functions

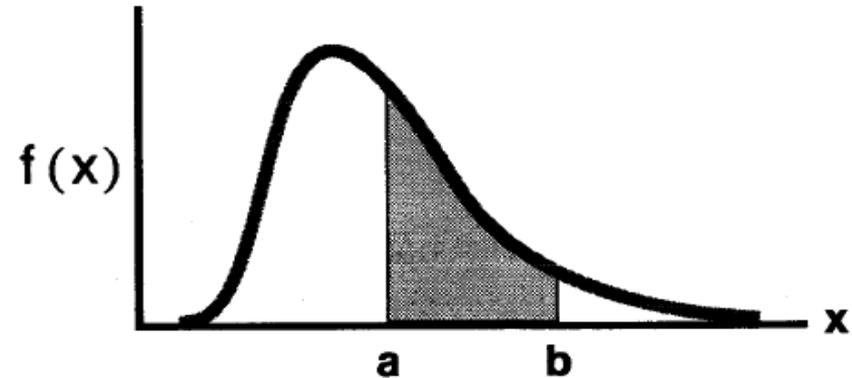
## • Continuous Probability Density

$f(x)$  = probability density function (PDF)

$$f(x) \geq 0$$

$$\text{Probability}\{a \leq x \leq b\} = \int_a^b f(x)dx$$

$$\text{Normalization: } \int_{-\infty}^{\infty} f(x)dx = 1$$



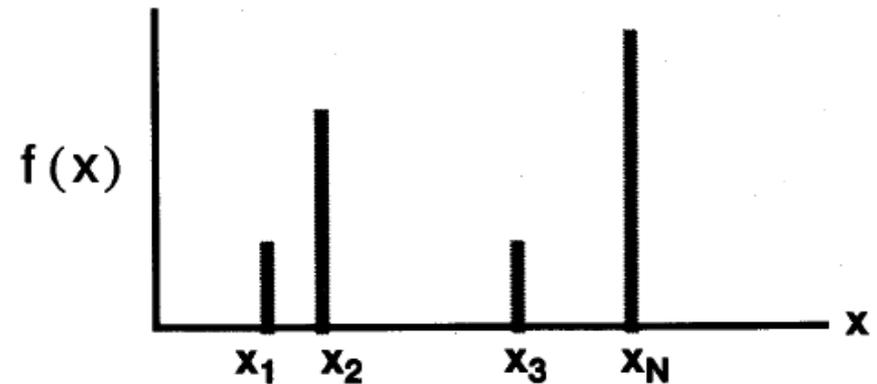
## • Discrete Probability Density

$\{f_k\}$ ,  $k = 1, \dots, N$ , where  $f_k = f(x_k)$

$$f_k \geq 0$$

$$\text{Probability}\{x = x_k\} = f_k$$

$$\text{Normalization: } \sum_{k=1}^N f_k = 1$$



# Continuous PDF & CDF

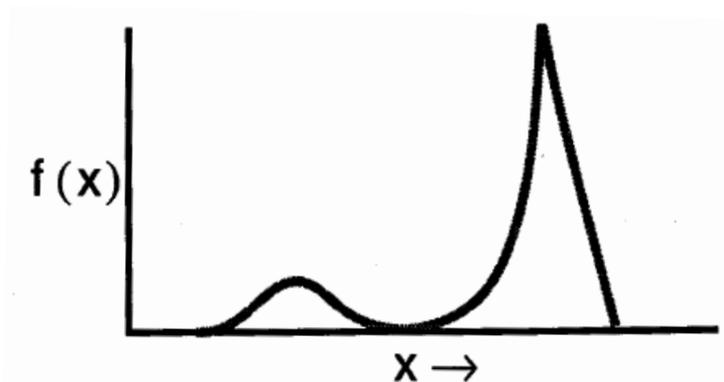
## • Probability Density Function (PDF)

$f(x)$  = probability density function (PDF)

$$f(x) \geq 0$$

$$\text{Probability}\{a \leq x \leq b\} = \int_a^b f(x)dx$$

$$\text{Normalization: } \int_{-\infty}^{\infty} f(x)dx = 1$$



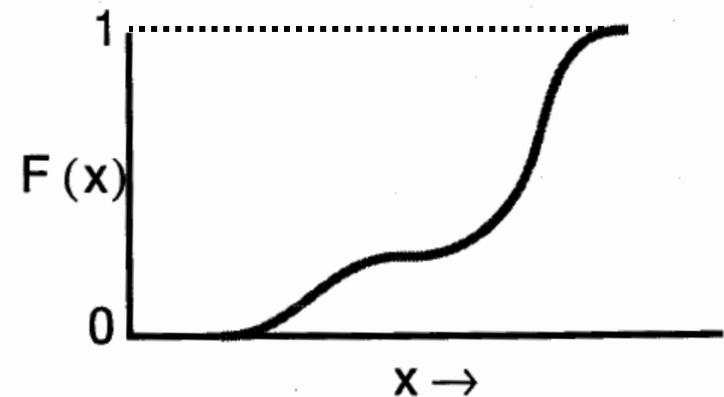
## • Cumulative Distribution Function (CDF)

$$F(x) = \int_{-\infty}^x f(x')dx'$$

$$0 \leq F(x) \leq 1$$

$$\frac{dF(x)}{dx} \geq 0$$

$$F(-\infty) = 0, \quad F(\infty) = 1$$



Note: convention is to use  $f(x)$  for PDF,  $F(x)$  for CDF

# Discrete PDF & CDF

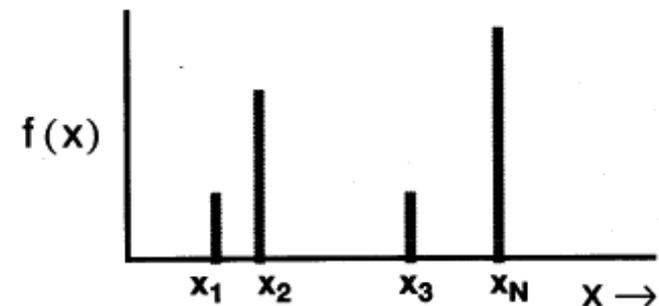
## Discrete PDF's

- Probability Density Function (PDF)

$\{f_k\}$ , where  $f_k = f(x_k)$ ,  $k=1, \dots, N$

$$f_k \geq 0$$

$$\sum_{j=1}^N f_j = 1$$



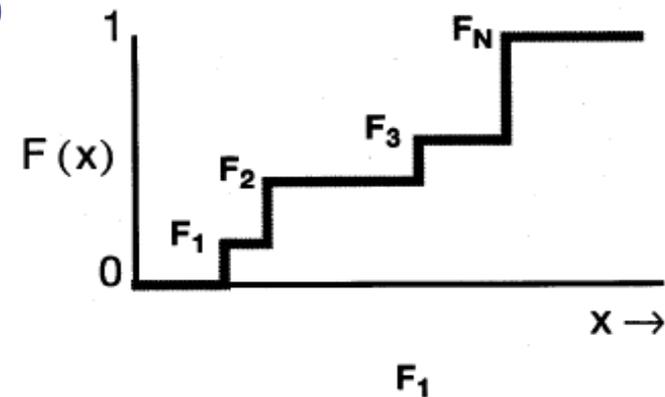
- Cumulative Distribution Function (CDF)

$\{F_k\}$ , where  $F_k = \sum_{j=1}^k f_j$ ,  $k=1, \dots, N-1$

and

$$F_0 = 0,$$

$$F_N = 1$$



Note: convention is to use  $f_j$  for PDF,  $F_j$  for CDF

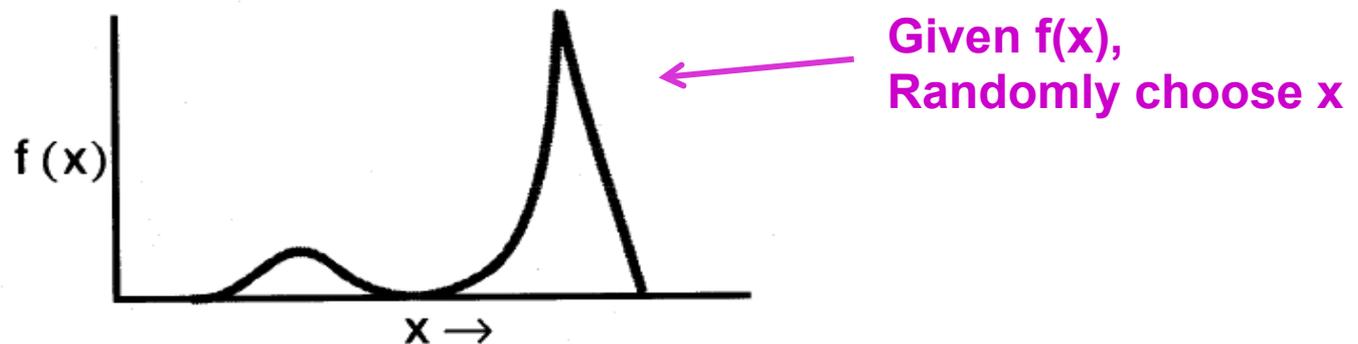
# Random Sampling

---

The key to Monte Carlo methods is the notion of *random sampling*.

- The problem can be stated this way:

Given a probability density,  $f(x)$ , produce a sequence of  $\hat{X}$ 's.  
The  $\hat{X}$ 's should be distributed in the same manner as  $f(x)$ .



- The use of random sampling distinguishes Monte Carlo from other methods
- When Monte Carlo is used to solve the integral Boltzmann transport equation:
  - Random sampling models the outcome of physical events (e.g., neutron collisions, fission process, sources, .....

# Random Sampling

---

- **Basic procedure for analytic random sampling**

- ① Convert PDF  $f(x)$  to CDF  $F(x)$

- ② Generate RN  $\xi$  on  $(0,1)$

- ③ Solve for  $x$ :  $F(x) = \xi$

If this is repeated many time, the resulting PDF will approach  $f(x)$

- **Formally**

– Solve for  $x$ :  $\xi = \int_{-\infty}^x f(y) dy$

– Or:  $x = F^{-1}(\xi)$

# ★★★★★ Direct Sampling ★★★★★

- Direct solution of  $x = F^{-1}(\xi)$

Solve for  $x$ : 
$$\xi = \int_{-\infty}^x f(y) dy$$

- **Sampling procedure**

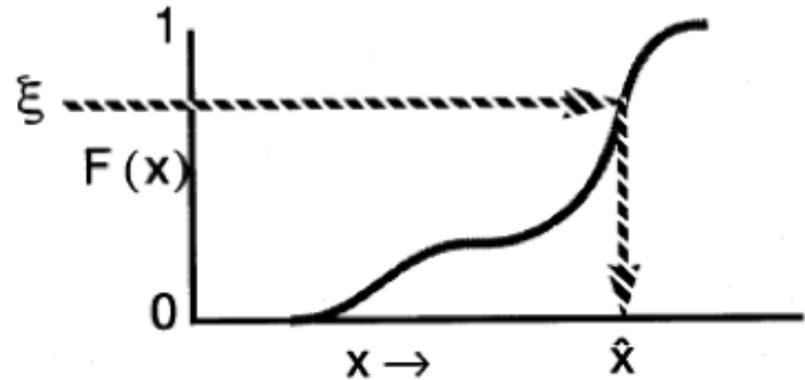
- Generate  $\xi$
- **Determine  $x$  such that  $F(x) = \xi$**

- **Advantages**

- Straightforward mathematics & coding
- "High-level" approach

- **Disadvantages**

- Often involves complicated functions
- In some cases,  $F(x)$  cannot be inverted (e.g., Klein-Nishina)

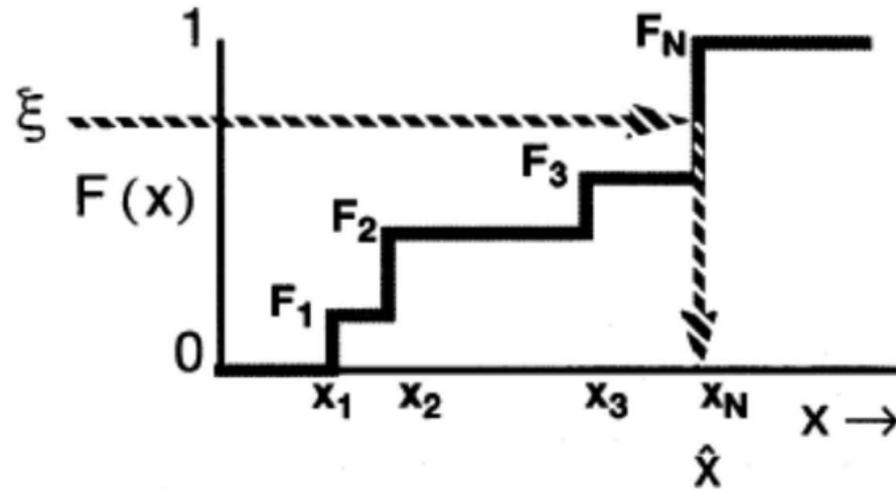


# Discrete PDFs

## • Sampling from Discrete PDF's - Conventional Procedure

Direct Solution of  $\mathbf{x}' \leftarrow F^{-1}(\xi)$

- (1) Generate  $\xi$
- (2) Determine  $k$  such that  $F_{k-1} \leq \xi < F_k$
- (3) Return  $\mathbf{x}' = x_k$



## • Step (2) requires a table search

- Linear table searches require  $O(N)$  time - use when  $N$  small
- Binary table searches require  $O(\ln N)$  time - use when  $N$  large
- **An alternative method – alias sampling – eliminates the table search & requires  $O(1)$  time, independent of  $N$**

## • For some discrete PDFs, $F_k$ 's are not precomputed.

- Use linear search, with  $F_k$ 's computed on-the-fly as needed

# Discrete Uniform PDF

- Example - Sampling from Discrete Uniform PDF

- Discrete Uniform PDF

- $f_k = 1 / N$ ,  $k = 1, \dots, N$
- $F_k = k / N$ ,  $F_0=0$ ,  $F_N=1$

- Sampling procedure:

- Could use table search method, ....
- Easier, for this special case:

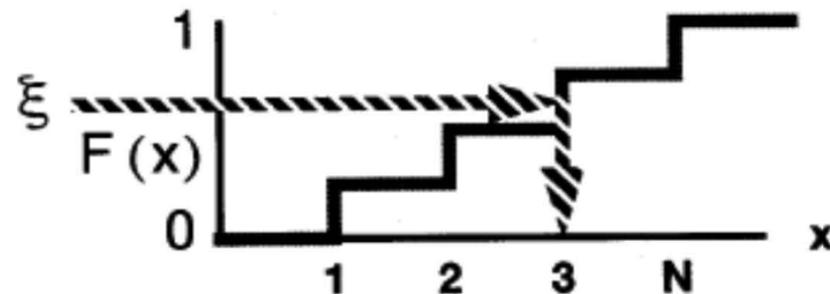
$$k \leftarrow 1 + \text{floor}( N \xi ),$$

floor(y) gives largest integer  $< y$

– Fortran:  $k = 1 + \text{int}( N * \text{rang}() )$

C:  $k = 1 + \text{floor}( N * \text{rang}() )$

– Note: must be sure that  $1 \leq k \leq N$



## Discrete PDFs - Examples

---

- Example – Pick 1 Powerball number, uniform integer in [1,69]

```
k = int( 1 + 69*rang() )
```

- Example - loaded die, faces show 2,2,3,4,5,5 – simulate 1 roll

```
pdf(1:6) = [ 0./6., 2./6., 1./6., 1./6., 2./6., 0./6. ]
cdf(1:6) = [ 0./6., 2./6., 3./6., 4./6., 6./6., 6./6. ]
```

```
r = rang()
do j = 1, 6
  if( r < cdf(j) ) then
    k = j
    exit
  endif
enddo
```

```
{result is k}
```

This coding is a simple linear search to determine an integer  $k$  in the range [1,6]

Search for the first occurrence of  $\xi \leq \text{cdf}(j)$

# Random Sampling -- Discrete PDFs

---

- **Multigroup Scattering**

- Scatter from group  $g$  to group  $g'$ , where  $1 \leq g' \leq G$

$$f_{g'} = \frac{\sigma_{g \rightarrow g'}}{\sum_{k=1}^G \sigma_{g \rightarrow k}}$$

- **Selection of scattering nuclide for a collision**

- $K$  = number of nuclides in composition

$$f_k = \frac{N^{(k)} \sigma_s^{(k)}}{\sum_{j=1}^K N^{(j)} \sigma_s^{(j)}}$$

# ★★★★★ Direct Sampling ★★★★★

- Direct solution of  $x = F^{-1}(\xi)$

Solve for  $x$ : 
$$\xi = \int_{-\infty}^x f(y) dy$$

- **Sampling procedure**

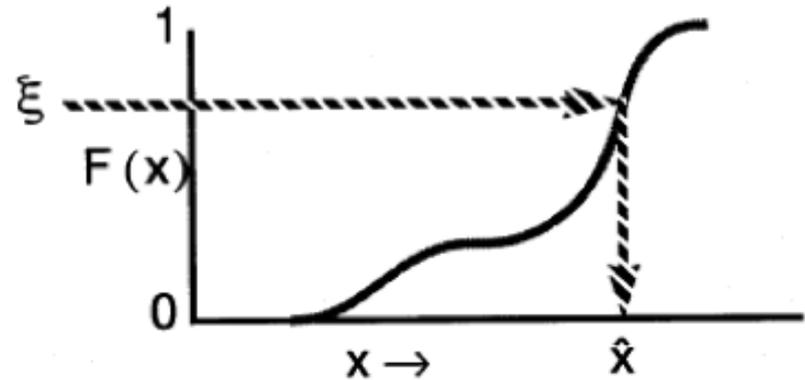
- Generate  $\xi$
- **Determine  $x$  such that  $F(x) = \xi$**

- **Advantages**

- Straightforward mathematics & coding
- "High-level" approach

- **Disadvantages**

- Often involves complicated functions
- In some cases,  $F(x)$  cannot be inverted (e.g., Klein-Nishina)



# Continuous PDFs - Exponential

---

## Examples - Sampling from an Exponential PDF

**PDF:**  $f(x) = \Sigma \cdot e^{-\Sigma x}, \quad x > 0$

**CDF:**  $F(x) = \int_0^x f(y) dy = \int_0^x \Sigma \cdot e^{-\Sigma y} dy = -e^{-\Sigma y} \Big|_0^x = 1 - e^{-\Sigma x}$

### Direct sampling:

**Solve for x:**  $F(x) = \xi$

Solving  $\xi = 1 - e^{-\Sigma x}$  gives:  $x \leftarrow -\ln(1 - \xi) / \Sigma$

or

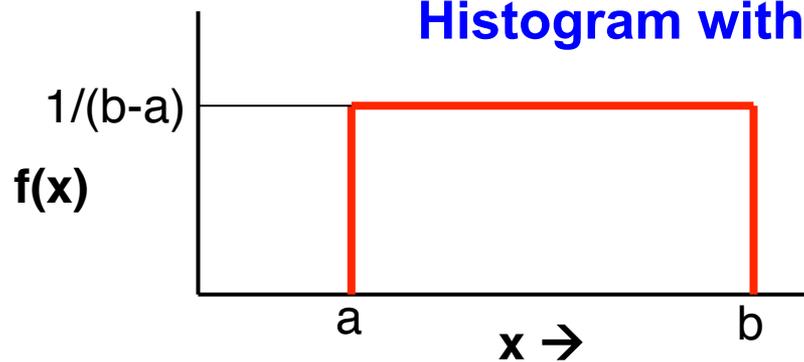
$$x \leftarrow -\ln \xi / \Sigma$$

Although  $(1 - \xi) \neq \xi$ ,  
both  $\xi$  and  $(1 - \xi)$  are uniformly distributed on  $(0,1)$ ,  
so that we can use either in the random sampling procedure.

i.e., the numbers are different, but the distributions are the same

# Continuous PDFs - Uniform

**Example - Sampling from uniform PDF in range (a,b),  
Histogram with 1 bin**



**PDF:**  $f(x) = 1/(b-a), \quad a \leq x \leq b$   
 $= 0 \quad x < a, \text{ or } x > b$

**CDF:**  $F(x) = (x-a)/(b-a), \quad a \leq x \leq b$

**Sampling scheme:**

$F(x) = \xi, \quad \text{solve for } x$   
 $(x-a)/(b-a) = \xi$

$x \leftarrow a + (b-a) \xi$

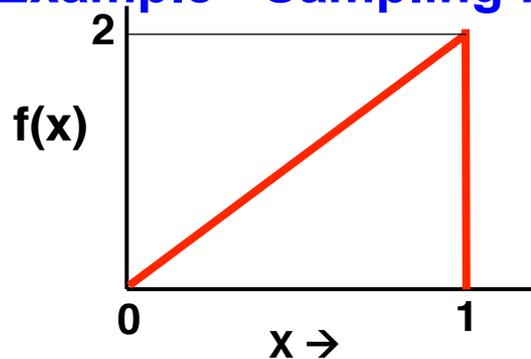
Note: Often implemented as:

$f = \xi$

$x \leftarrow (1-f) a + f b$

# Continuous PDFs – Linear (1)

Example - Sampling from an increasing linear PDF in range [0,1]



**PDF:**  $f(x) = 2x, \quad 0 \leq x \leq 1$

**CDF:**  $F(x) = x^2, \quad 0 \leq x \leq 1$

**Sampling scheme:**

$F(x) = \xi, \quad \text{solve for } x$

$x^2 = \xi$

$x \leftarrow \text{sqrt}(\xi)$

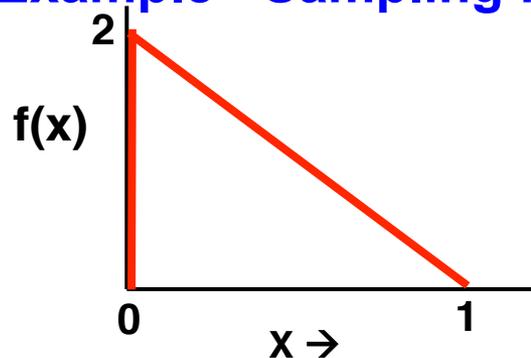
While not obvious, 2 alternative schemes for sampling  $x$  are:

$x \leftarrow \max(\xi_1, \xi_2)$

$x \leftarrow 1 - \text{abs}(\xi_1 - \xi_2)$

## Continuous PDFs – Linear (2)

Example - Sampling from a decreasing linear PDF in range [0,1]



**PDF:**  $f(x) = 2 - 2x, \quad 0 \leq x \leq 1$

**CDF:**  $F(x) = 2x - x^2, \quad 0 \leq x \leq 1$

**Sampling scheme:**

$$F(x) = \xi, \quad \text{solve for } x$$

$$2x - x^2 = \xi$$

$$x^2 - 2x + 1 = 1 - \xi$$

$$(x-1)^2 = 1 - \xi$$

$$x - 1 = \pm \sqrt{1 - \xi}$$

Choose the minus sign for correct range in x:

$$x \leftarrow 1 - \sqrt{1 - \xi}$$

Or, since  $\xi$  and  $1 - \xi$  have the same distribution:

$$x \leftarrow 1 - \sqrt{\xi}$$

# Continuous PDFs – Power Law on [0,1]

## Example - Sampling from power law PDF in range [0,1],

Note:  $(n+1)$  is necessary, so that  $\int_0^{\infty} f(x') dx' = 1$

**PDF:**  $f(x) = (n+1) x^n, \quad n > 0, \quad 0 \leq x \leq 1$   
 $= 0 \quad \quad \quad x < 0, \text{ or } x > 1$

**CDF:**  $F(x) = \int_0^x f(y) dy = \int_0^x (n+1) \cdot y^n dy = (n+1) \cdot \frac{y^{n+1}}{n+1} \Big|_0^x = x^{n+1}, \quad 0 \leq x \leq 1$

**Sampling scheme:**  $F(x) = \xi, \quad \text{solve for } x$

$$x^{n+1} = \xi$$

$$x \leftarrow \xi^{1/(n+1)}$$

For power laws on [0,1]:

$n=1:$   $f(x) = 2x, \quad F(x) = x^2, \quad x \leftarrow \sqrt{\xi}$

$n=2:$   $f(x) = 3x^2, \quad F(x) = x^3, \quad x \leftarrow \sqrt[3]{\xi}$

$n=3:$   $f(x) = 4x^3, \quad F(x) = x^4, \quad x \leftarrow \sqrt[4]{\xi}$

# Direct Sampling – Common PDFs

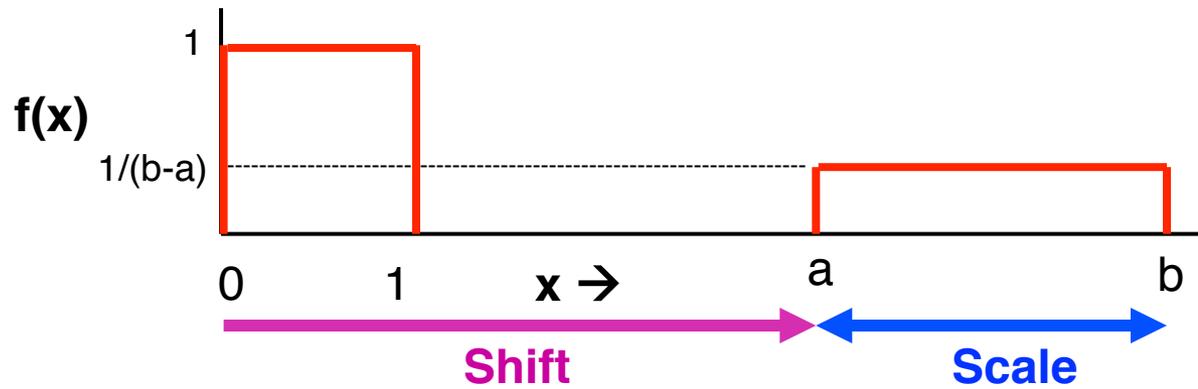
<i>Probability Density Function</i>		<i>Direct Sampling Method</i>
<b>Linear:</b>	$f(x) = 2x, \quad 0 < x < 1$	$x \leftarrow \sqrt{\xi}$
<b>Exponential:</b>	$f(x) = e^{-x}, \quad 0 < x$	$x \leftarrow -\log \xi$
<b>2D Isotropic:</b>	$f(\vec{p}) = \frac{1}{2\pi}, \quad \vec{p} = (u, v)$	$u \leftarrow \cos 2\pi \xi_1$ $v \leftarrow \sin 2\pi \xi_1$
<b>3D Isotropic:</b>	$f(\vec{\Omega}) = \frac{1}{4\pi}, \quad \vec{\Omega} = (u, v, w)$	$u \leftarrow 2\xi_1 - 1$ $v \leftarrow \sqrt{1-u^2} \cos 2\pi \xi_2$ $w \leftarrow \sqrt{1-u^2} \sin 2\pi \xi_2$
<b>Maxwellian:</b>	$f(x) = \frac{2}{T\sqrt{\pi}} \sqrt{\frac{x}{T}} e^{-x/T}, \quad 0 < x$	$x \leftarrow T \left( -\log \xi_1 - \log \xi_2 \cos^2 \frac{\pi \xi_3}{2} \right)$
<b>Watt Spectrum:</b>	$f(x) = \frac{2e^{-ab/4}}{\sqrt{\pi a^3 b}} e^{-x/a} \sinh \sqrt{bx}, \quad 0 < x$	$w \leftarrow a \left( -\log \xi_1 - \log \xi_2 \cos^2 \frac{\pi \xi_3}{2} \right)$ $x \leftarrow w + \frac{a^2 b}{4} + (2\xi_4 - 1) \sqrt{a^2 b w}$
<b>Normal:</b>	$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$	$x \leftarrow \mu + \sigma \sqrt{-2\log \xi_1} \cos 2\pi \xi_2$

---

# Linear Transformations & Scaling

# Continuous PDFs - Uniform

## Example – Shifting & Scaling a 1-bin Histogram



**PDF:**

$$f(x) = 1$$

$$f(x) = 1/(b-a)$$

**CDF:**

$$F(x) = x$$

$$F(x) = (x-a)/(b-a)$$

**Range:**

$$[ 0, 1 ]$$

$$[ a, b ]$$

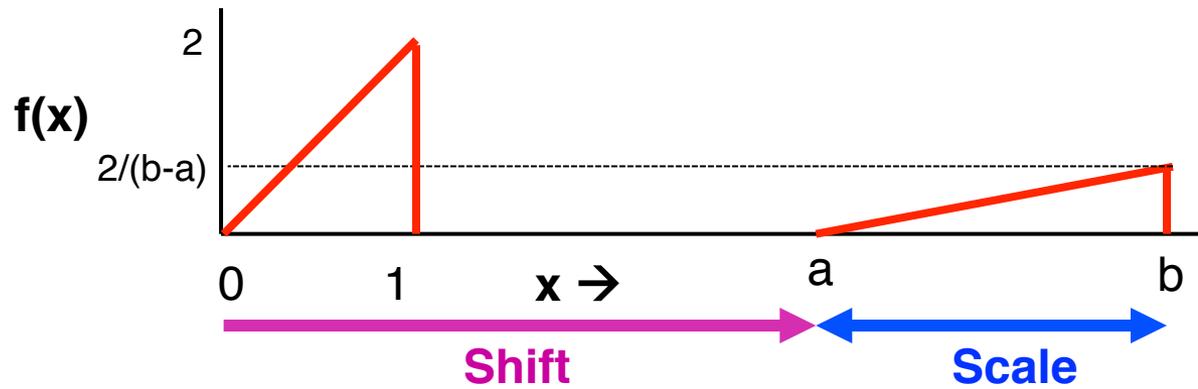
**Sampling:**

$$x \leftarrow \xi$$

$$x \leftarrow a + (b-a) \xi$$

# Continuous PDFs - Linear

## Example – Shifting & Scaling a unit linear PDF



**PDF:**

$$f(x) = 2x$$

$$f(x) = 2(x-a)/(b-a)^2$$

**CDF:**

$$F(x) = x^2$$

$$F(x) = [(x-a)/(b-a)]^2$$

**Range:**

$$[0, 1]$$

$$[a, b]$$

**Sampling:**

$$x \leftarrow \sqrt{\xi}$$

$$x \leftarrow a + (b-a)\sqrt{\xi}$$

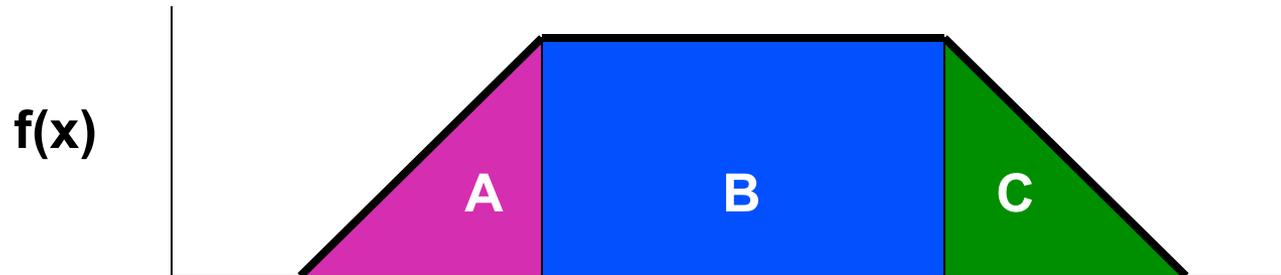
---

# Composition Methods

# Composition Method

---

- A complicated PDF . . .



- . . . Can be decomposed into a sum of simpler PDFs

$$f(x) = p_A f_A(x) + p_B f_B(x) + p_C f_C(x)$$

where  $p_A + p_B + p_C = 1$

and each piece of the PDF is scaled s.t. area is 1

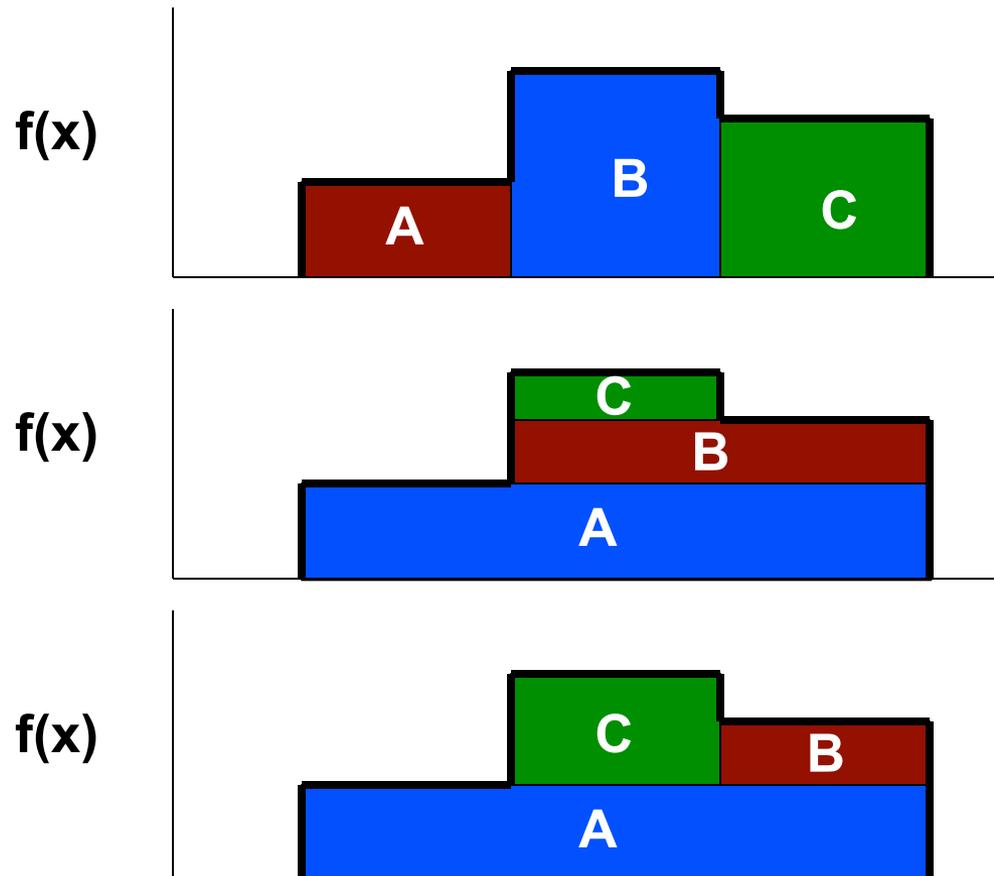
- Sampling then proceeds in 2 steps:

- ① Discrete sampling from  $\{p_A, p_B, p_C\}$  to select A, B, or C
- ② Continuous sampling within the chosen PDF piece

# Composition Method

---

- A PDF can be decomposed in many different ways . . .



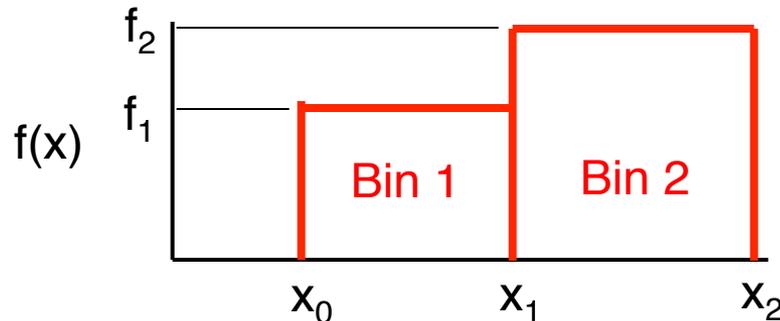
$$f(x) = p_A f_A(x) + p_B f_B(x) + p_C f_C(x)$$

# Continuous PDFs - Histogram

## Example - Sampling from histogram with 2 bins

$$A_1 = (x_1 - x_0) \cdot f_1$$

$$A_2 = (x_2 - x_1) \cdot f_2$$



$$p_1 = \text{Prob}\{ x_0 < x < x_1 \} = A_1 / (A_1 + A_2)$$

$$p_2 = \text{Prob}\{ x_1 < x < x_2 \} = A_2 / (A_1 + A_2)$$

$$p_1 + p_2 = 1$$

### Two-step sampling procedure:

1. Select a bin,  $b$ :

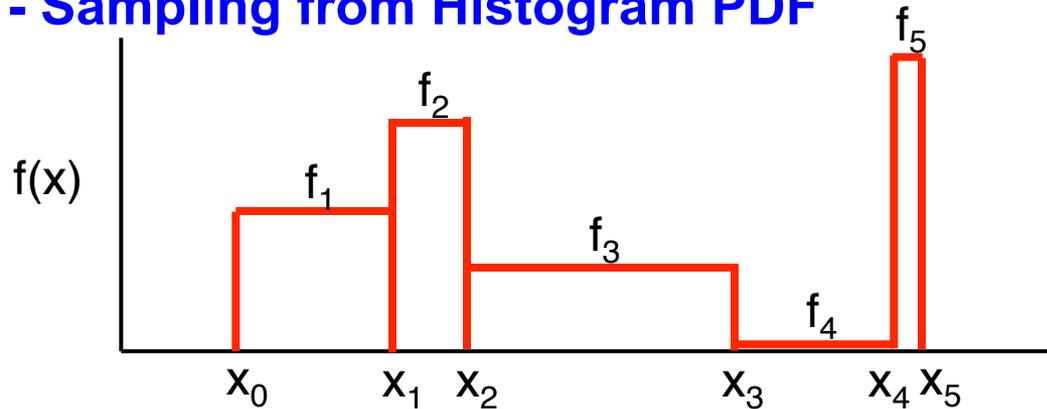
If  $\xi_1 < p_1$ , select  $b = \text{bin 1}$   
 otherwise, select  $b = \text{bin 2}$

2. Sample  $x$  within bin:

$$x \leftarrow x_{b-1} + \xi_2 \cdot (x_b - x_{b-1})$$

# Continuous PDFs - Histograms

## Example - Sampling from Histogram PDF



**Two-step sampling:**

- (1) Sample from discrete PDF to select a bin
- (2) Sample from uniform PDF within bin

• **Discrete PDF:**  $p_k = f_k \cdot (x_k - x_{k-1}), \quad k = 1, \dots, N, \quad \sum p_k = 1$

– Generate  $\xi_1$

– Use table search to select  $k$

• **Uniform sampling within bin  $k$**

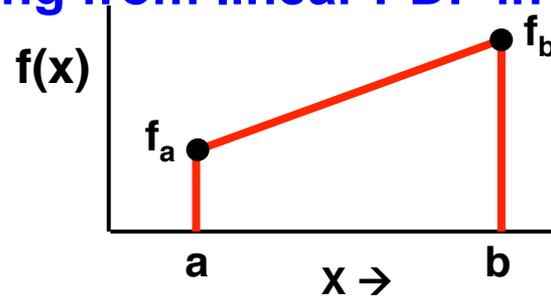
– Generate  $\xi_2$

– Then,

$$x \leftarrow x_{k-1} + (x_k - x_{k-1}) \cdot \xi_2$$

# Continuous PDFs – Linear

**Example - Sampling from linear PDF in range  $[a,b]$ , 1 bin**



**PDF:**  $f(x) = f_a + m(x-a), \quad m = (f_b - f_a)/(b-a), \quad a \leq x \leq b$

**CDF:** 
$$F(x) = (m/2)x^2 + (f_a - ma)x + (ma^2/2 - f_a a)$$

$$= Ax^2 + Bx + C$$

**Sampling scheme:**  $F(x) = \xi, \quad \text{solve for } x$   

$$x = \{ -B \pm \sqrt{B^2 - 4A(C-\xi)} \} / 2A$$

- ➔ Awfully complicated, and sensitive to numerical roundoff
- ➔ There must be a simpler scheme (there is ...)

# Continuous PDFs – Linear

## Example - Sampling from linear PDF in range $[a,b]$ , 1 bin

### Composition method #1

Decompose the original PDF into the sum of 2 PDFs, uniform + linear:

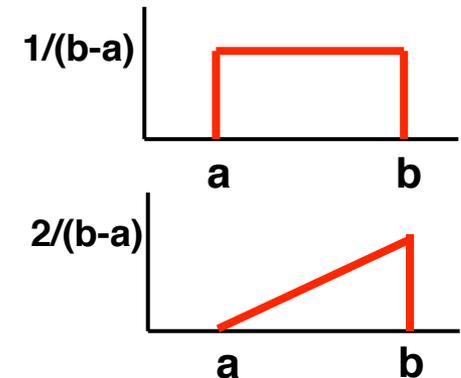
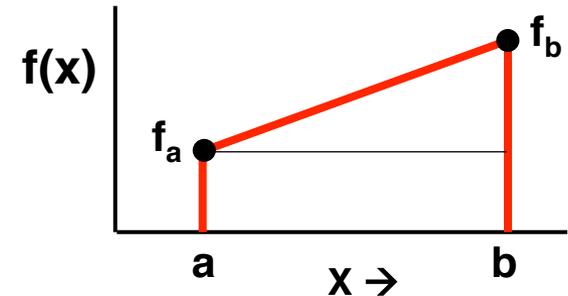
$$f(x) = p_u u(x) + p_l l(x)$$

$$u(x) = \text{uniform on } a \leq x \leq b,$$

$$p_u = \{ \min(f_a, f_b) (b-a) \} / \{ .5(f_a + f_b) (b-a) \}$$

$$l(x) = \text{linear on } a \leq x \leq b,$$

$$p_l = \{ .5 \text{ abs}(f_b - f_a) (b-a) \} / \{ .5(f_a + f_b) (b-a) \}$$



**Sampling scheme:**

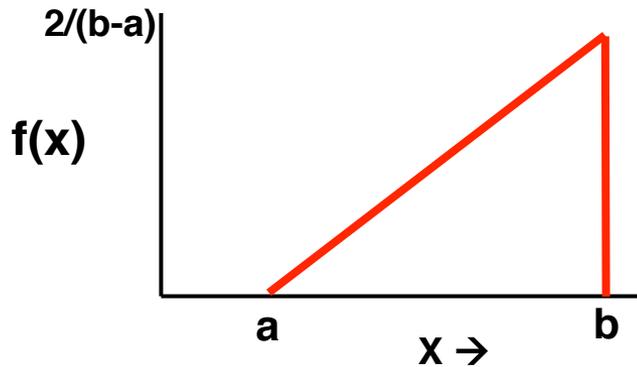
```

if(  $\xi_1 < p_u$  )
     $x \leftarrow a + (b-a) \xi_2$ 
else
    if(  $f_b > f_a$  )     $x \leftarrow a + (b-a) \text{sqrt}(\xi_2)$ 
    else                 $x \leftarrow a + (b-a) (1 - \text{sqrt}(\xi_2))$ 

```

# Continuous PDFs – Linear

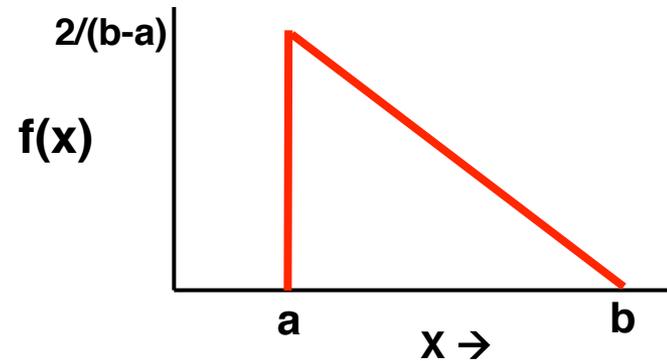
## Increasing linear PDF



Random sampling can be done with a simple shifting & scaling of the unit PDF:

$$x \leftarrow a + (b-a) \text{sqrt}(\xi)$$

## Decreasing linear PDF



Random sampling can be done with a simple shifting & scaling of the unit PDF:

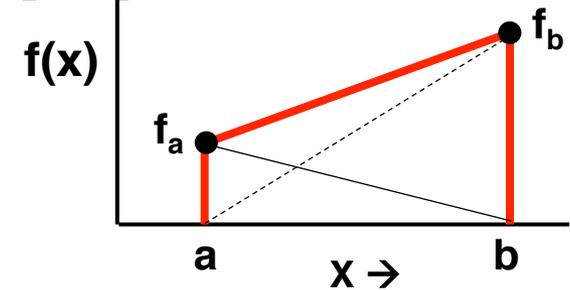
$$x \leftarrow a + (b-a) (1 - \text{sqrt}(\xi))$$

# Continuous PDFs – Linear

## Example - Sampling from linear PDF in range $[a,b]$ , 1 bin

### Composition method #2

Decompose the original PDF into the sum of 2 PDFs, increasing + decreasing linear:



$$f(x) = p_m m(x) + p_l l(x)$$

$m(x)$  = linear decreasing on  $a \leq x \leq b$ ,

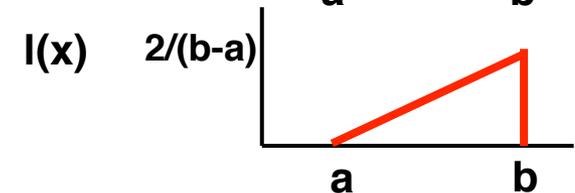
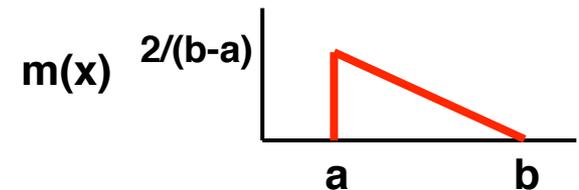
$$p_m = \{ .5 f_a (b-a) \} / \{ .5(f_a+f_b) (b-a) \}$$

$$= f_a / (f_a+f_b)$$

$l(x)$  = linear increasing on  $a \leq x \leq b$ ,

$$p_l = \{ .5 f_b (b-a) \} / \{ .5(f_a+f_b) (b-a) \}$$

$$= f_b / (f_a+f_b)$$



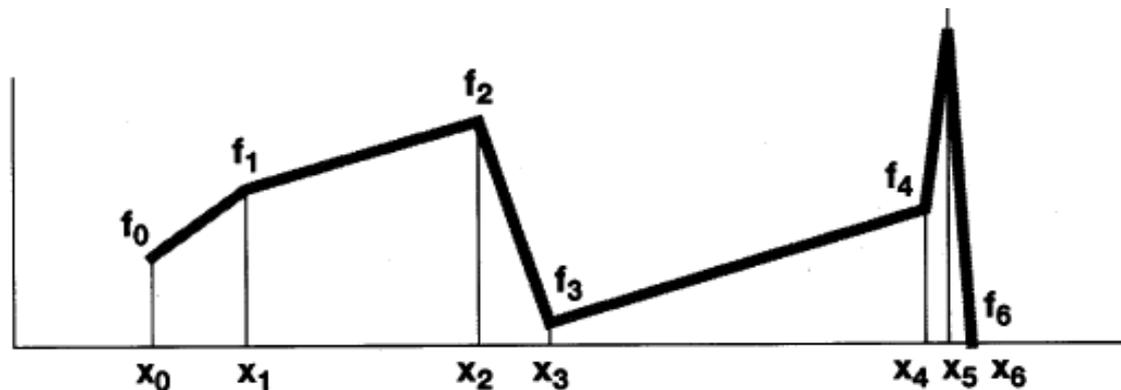
Sampling scheme: if(  $\xi_1 < p_l$  )

$$x \leftarrow a + (b-a) \text{sqrt}(\xi_2)$$

else

$$x \leftarrow a + (b-a) (1 - \text{sqrt}(\xi_2))$$

# Continuous PDFS – Piecewise Linear



**Two-step sampling:**

- (1) **Sample from discrete PDF to select a bin**
- (2) **Sample from linear PDF within bin**

• Discrete PDF: 
$$p_k = \frac{(f_k + f_{k-1})}{2} \cdot (x_k - x_{k-1}), \quad k = 1, \dots, N$$

— generate  $\xi$

— use table search or alias method to select  $\mathbf{K}$

• Linear sampling within bin  $\mathbf{K}$ :

— generate  $\xi$

— then,

$$\text{if } \xi_1 < \frac{f_{k-1}}{f_k + f_{k-1}}, \quad \hat{x} \leftarrow x_k - (x_k - x_{k-1}) \sqrt{\xi_2}$$

otherwise

$$\hat{x} \leftarrow x_{k-1} + (x_k - x_{k-1}) \sqrt{\xi_2}$$

# Rejection Methods

# Rejection Sampling

- **Von Neumann:**

" ..... *it seems objectionable to compute a transcendental function of a random number.* "

- **Select a bounding function,  $g(x)$ , such that**

- $c \geq g(x) > f(x)$  for all  $x$
- $g(x)$  is an easy-to-sample PDF

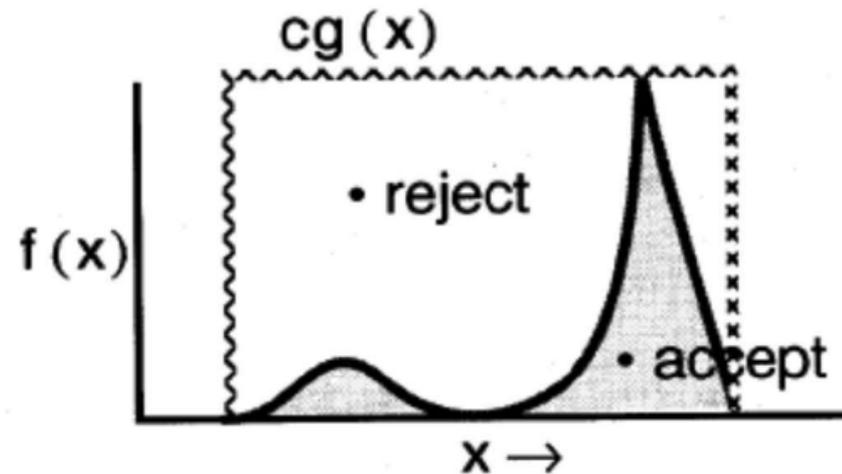
- **Sampling Procedure:**

- sample  $x'$  from  $g(x)$ :  $x' \leftarrow G^{-1}(\xi_1)$

- test:  $\xi_2 \leq c g(x') < f(x')$

if **true**  $\rightarrow$  accept  $x'$ , done

if **false**  $\rightarrow$  reject  $x'$ , try again



- **Advantages**

- Simple computer operations

- **Disadvantages**

- “Low-level” approach, sometimes hard to understand

# Rejection Sampling - Examples

- Sample from a PDF

$$f(x) = c \cdot \text{erf}(x), \quad 0 \leq x \leq 5.$$

note:  $\text{erf}(\infty) = 1.$

Do

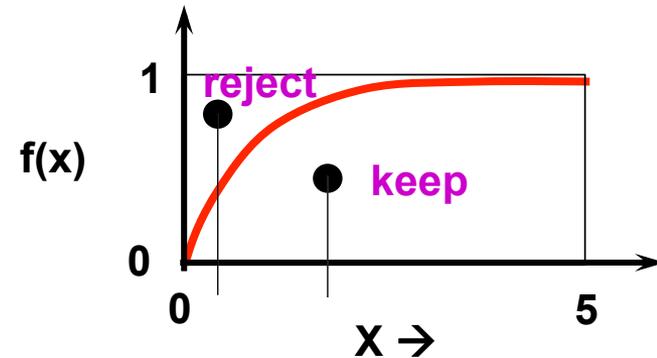
```
xtry = 5.*rang()
```

```
ftry = 1.*rang()
```

```
if( ftry <= erf(xtry) ) exit
```

Enddo

```
x = xtry
```



- Select  $(x,y)$  points uniformly in a disk

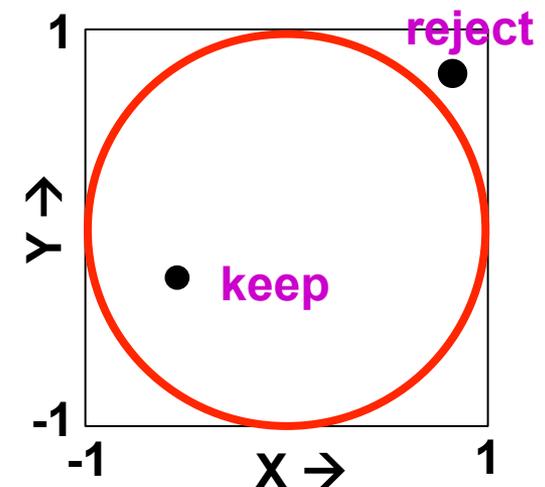
Do

```
x = 2.*rang() - 1.
```

```
y = 2.*rang() - 1.
```

```
if( x**2 + y**2 < 1.0 ) exit
```

Enddo



# Direct vs. Rejection - 2D Direction Cosines

## Example — 2D Isotropic

$$f(\vec{p}) = \frac{1}{2\pi}, \quad \vec{p} = (u, v)$$

### Rejection *(old vim)*

```

SUBROUTINE AZIRN_VIM( S, C )
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
100 R1=2.*RANF() - 1.
    R1SQ=R1*R1
    R2=RANF()
    R2SQ=R2*R2
    RSQ=R1SQ+R2SQ
    IF(1.-RSQ)100,105,105
105 S=2.*R1*R2/RSQ
    C=(R2SQ-R1SQ)/RSQ
RETURN
END

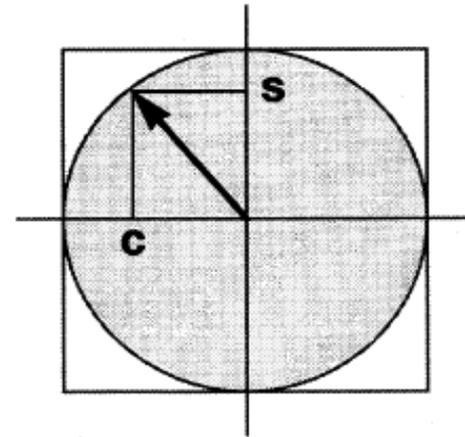
```

### Direct *(racer, new vim)*

```

subroutine azirn_new( s, c )
implicit double precision (a-h,o-z)
parameter ( twopi = 2.*3.14159265 )
phi = twopi*ranf()
c = cos(phi)
s = sin(phi)
return
end

```



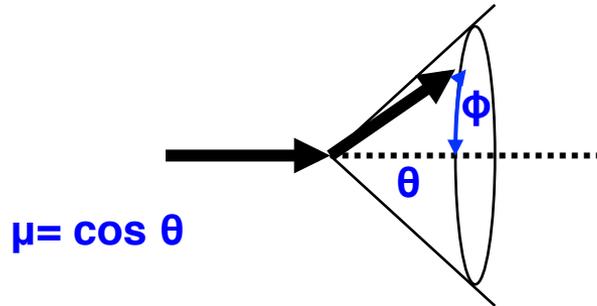
## Isotropic Scatter – Sampling the Scattering Angle

- **Consider isotropic scattering**

- Any direction is equally likely

- Interpret as:

"pick a random point on a unit sphere, then get direction-cosines"



- **Rejection method for scatter angle sampling**

- Pick  $x, y, z$  randomly in unit cube

- If  $x, y, z$  outside unit sphere, reject and try again

- If  $x, y, z$  inside unit sphere, scale so that  $x^2 + y^2 + z^2 = 1$

- Get direction-cosines of angles,  $u, v, w$

- **Direct method for scatter angle sampling**

$$f(\hat{\Omega}) = \frac{1}{4\pi}, \quad \frac{d\hat{\Omega}}{4\pi} = \frac{\sin\theta \cdot d\theta}{2} \cdot \frac{d\phi}{2\pi}$$

$$f(\theta, \phi) = \frac{\sin\theta \cdot d\theta}{2} \cdot \frac{d\phi}{2\pi}, \quad 0 \leq \theta \leq \pi, \quad 0 \leq \phi \leq 2\pi$$

$$f(\theta) = \int_0^{2\pi} f(\theta, \phi) d\phi = \frac{\sin\theta}{2}$$

$$\mu = \cos\theta, \quad d\mu = -\sin\theta \cdot d\theta, \quad -1 \leq \mu \leq +1$$

$$f(\mu) = f(\theta) \left| \frac{d\theta}{d\mu} \right| = \frac{\sin\theta}{2} \cdot \frac{1}{\sin\theta} = \frac{1}{2}$$

→  $\mu$  is distributed uniformly in  $[-1, 1]$

→  $\phi$  is distributed uniformly in  $[0, 2\pi]$

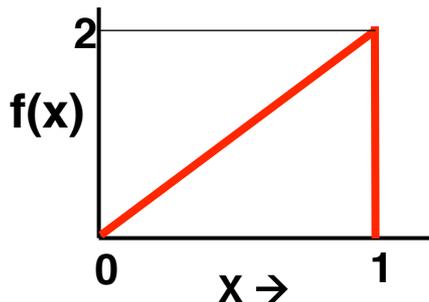
$$\mu \leftarrow 2\xi_1 - 1$$

$$\phi \leftarrow \xi_2 \cdot 2\pi$$

# Miscellaneous

## Continuous PDFs – Linear (8)

We have seen that a simple, increasing linear PDF in the range  $[0,1]$  can be sampled directly by inverting the CDF to obtain:



$$\text{PDF: } f(x) = 2x, \quad 0 \leq x \leq 1$$

$$\text{CDF: } F(x) = x^2, \quad 0 \leq x \leq 1$$

Sampling scheme:

$$F(x) = \xi, \quad \text{solve for } x$$

$$x \leftarrow \text{sqrt}(\xi)$$

While not obvious, some other schemes for sampling  $x$  are:

$$x = \xi_1$$

$$r = \xi_2$$

$$\text{if}( r > x ) \quad x = r$$

$$x \leftarrow \max(\xi_1, \xi_2)$$

$$x \leftarrow 1 - \text{abs}(\xi_1 - \xi_2)$$

Why consider these other schemes?

Sqrt() function used to be very expensive. The other schemes involve only simple non-arithmetic operations & were much faster.

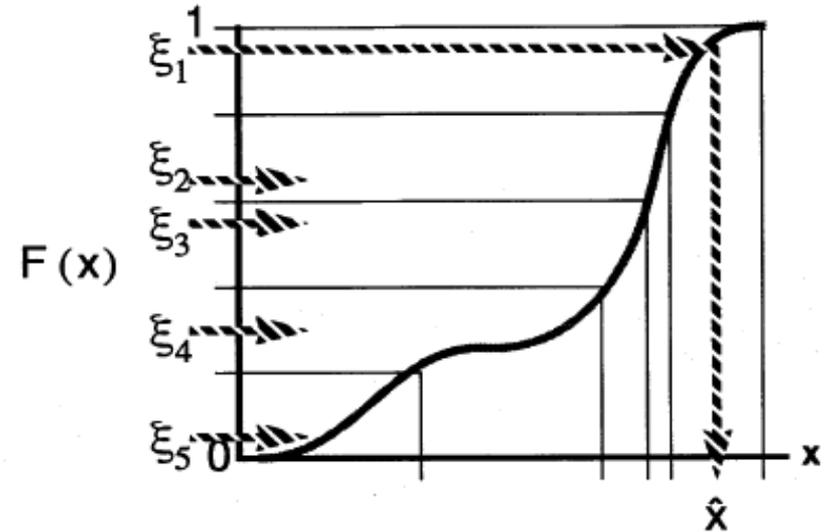
Today, sqrt() operations & computers are very fast – sqrt() is as fast as generating a 2<sup>nd</sup> RN. We usually go with the more obvious direct method.

BUT, the older schemes are still commonly used in production MC codes. Learn to recognize them.

# Stratified Sampling

If a specific number of samples,  $M$ , is needed from a single distribution:

- Naive approach — repeat the sampling procedure  $M$  times
- **Stratified sampling** approach
  - partition the sample space into  $M$  disjoint regions of **equal probability**
  - produce 1 sample from each region
- Stratified sampling considerations
  - $F(x)$  must be known & easy to partition
  - The number of partitions,  $M$ , must be known in advance
  - Must be relatively easy to sample **within** each given partition
  - Stratification improves the "coverage"
  - Stratified sampling reduces variance, at little or no computing cost



# Random Sampling -- References

---

Every Monte Carlo code developer who works with random sampling should own & read these references:

- D. E. Knuth, The Art of Computer Programming, Vol. 2: Semi-numerical Algorithms, 3<sup>rd</sup> Edition, Addison-Wesley, Reading, MA (1998).
- L. Devroye, Non-Uniform Random Variate Generation, Springer-Verlag, NY (1986).
- J. von Neumann, "Various Techniques Used in Conjunction with Random Digits," *J. Res. Nat. Bur. Stand. Appl. Math Series* 3, 36-38 (1951).
- C. J. Everett and E. D. Cashwell, "A Third Monte Carlo Sampler," LA9721-MS, Los Alamos National Laboratory, Los Alamos, NM (1983).
- H. Kahn, "Applications of Monte Carlo," AECU-3259, Rand Corporation, Santa Monica, CA (1954).

**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Optimal Random Sampling from Piecewise Linear Probability Density Functions**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



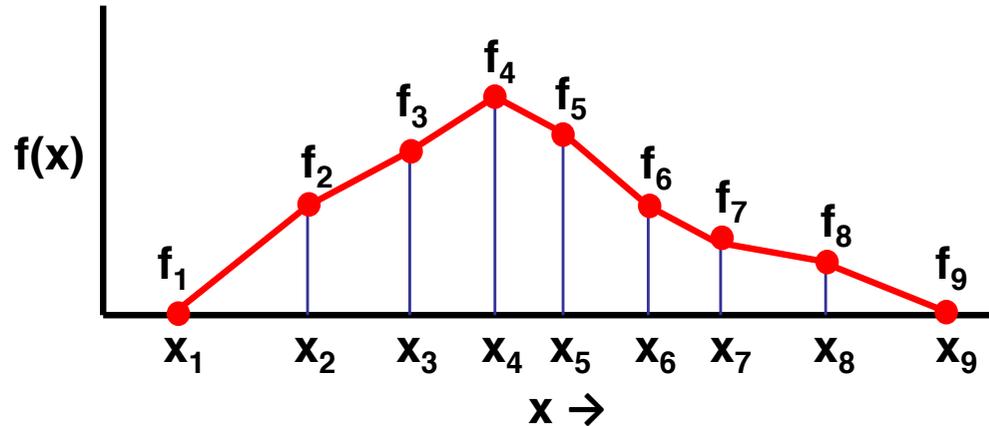
# Outline

---

- **Introduction**
- **Piecewise-Linear PDFs**
- **Conventional Approach to Random Sampling**
  - **Binary search in CDF**
  - **Direct inversion of iin PDF**
- **Optimal Approach**
  - **Alias sampling for CDF**
  - **Composition method for bin PDF**

# Introduction

- **Continuous Probability Density Functions (PDFs) are frequently approximated by tabulated piecewise-linear PDFs**



- Bin widths can be chosen adaptively to minimize relative error
- **This lecture addresses cases where the PDFs are**
  - Known at problem setup, prior to running any particle histories
  - Small to moderate number of entries, so that preprocessing & some extra storage is practical
  - Sampled often-enough during particle histories that any preprocessing time is unimportant
  - Generally most useful for PDFs found in source sampling & collision physics (exit energy & angle)

---

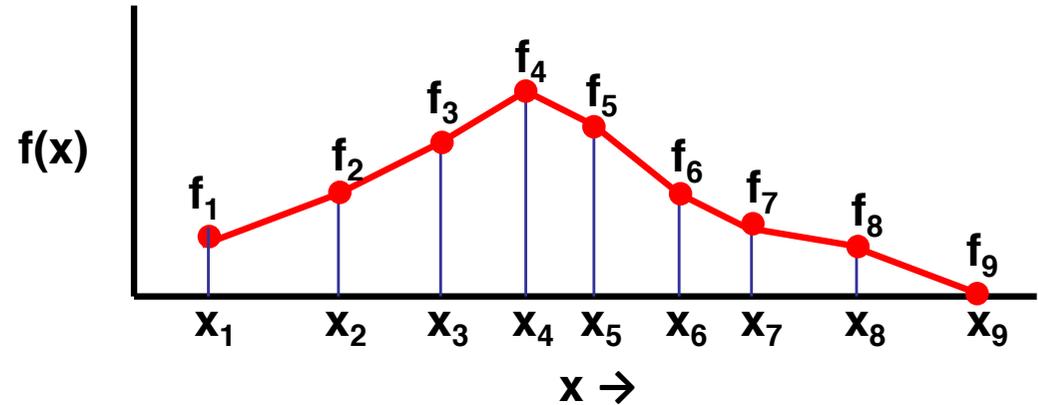
# Conventional Approach to Random Sampling from a Piecewise Linear PDF

# Piecewise Linear PDFs

- Many PDFs are represented as tabulated piecewise linear functions of  $E, \mu, x, \dots$

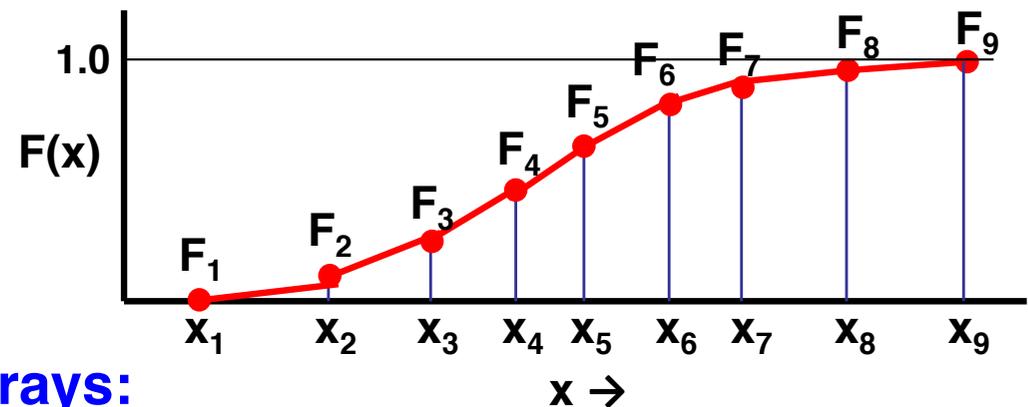
- Probability Density Function (PDF),  $f(x)$

N points, N-1 bins



- Cumulative Distribution Function (CDF),  $F(x)$

Quadratic shape within bins



- Usually stored as linear arrays:

$$x(1..N) = [ x_1, x_2, \dots, x_N ]$$

$$f(1..N) = [ f_1, f_2, \dots, f_N ]$$

$$F(1..N) = [ F_1, F_2, \dots, F_N ]$$

# Conventional Sampling Technique

---

- Data:**

$$\mathbf{x}(1..N) = [ \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N ]$$

$$\mathbf{f}(1..N) = [ \mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_N ]$$

$$\mathbf{F}(1..N) = [ \mathbf{F}_1, \mathbf{F}_2, \dots, \mathbf{F}_N ]$$

← computed at problem setup

- Two steps are required:**

1. Randomly sample a bin,  $k$

- $r = \xi$
- Search the CDF array to find the bin  $k$  containing  $r$ ,  

$$\mathbf{F}_k \leq r \leq \mathbf{F}_{k+1}, \quad 1 \leq k \leq N-1$$

2. Sample  $x'$  from the linear PDF within bin  $k$

Linear PDF from  $(x_k, f_k)$  to  $(x_{k+1}, f_{k+1}) \rightarrow$  Quadratic CDF,  $F(x)$

- $r = \xi$
- Solve for  $x'$ :  $r = F(x')$

# Search Algorithms

---

- **There is extensive literature on search algorithms**
  - D.E. Knuth, The Art of Computer Programming Vol 3 - Sorting & Searching
  - Many other references - books & journals
- **For general Monte Carlo codes with cross-section data, the commonly-used methods are linear search &/or binary search**
  - Linear search takes  $O(N)$  time, best when  $N \sim 10$  or less
  - Binary search takes  $O(\ln N)$  time, best when  $N \sim$  large
  - Linear searches are easier to program, less prone to code errors
  - For both linear & binary searches, need to consider what to do if  $x < x_1$  or  $x > x_N$  (x outside table)
    - Best to avoid this
    - Error stop? Use endpoint? Extrapolate?

# Binary Search to Sample Bin

- Given  $x$  and data table:  $N$ ,  $\text{table}(1..N)$
- Find  $k$  such that:  $\text{table}_k \leq \text{target} \leq \text{table}_{k+1}$ ,  $1 \leq k \leq N-1$

```

int binary_search( int* n,
                  double* table,
                  double* target )
{
    int jfirst, jlast, jmid;
    jfirst = 0;
    jlast = *n - 1;

    for(;;) {
        if( jlast-jfirst == 1 ) break;
        jmid = (jfirst+jlast)/2;
        if( *target >= table[jmid] ) {
            jfirst = jmid;
        }
        else {
            jlast = jmid;
        }
    }
    return jfirst+1;
}

```

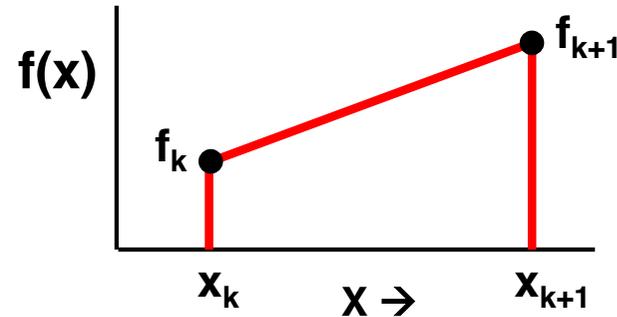
For use in random sampling, *target* is an RN in (0,1) so that an error check on out-of-range is not needed.

Not obvious, but:

- Guaranteed to terminate
- Guaranteed result in [ 1, N-1 ]

# Continuous PDFs – Linear

## Example - Sampling from linear PDF bin k



Note: normalized such that  $0.5 \cdot (f_k + f_{k+1}) \cdot (x_{k+1} - x_k) = 1$

**PDF:**  $f(x) = f_k + m(x - x_k), \quad m = (f_{k+1} - f_k) / (x_{k+1} - x_k), \quad x_k \leq x \leq x_{k+1}$

**CDF:** 
$$F(x) = (m/2)x^2 + (f_k - mx_k)x + (mx_k^2/2 - f_k x_k)$$

$$= A x^2 + B x + C$$

**Sampling scheme:**  $F(x) = \xi, \quad \text{solve for } x$   

$$x = \{ -B + \sqrt{B^2 - 4A[C - \xi]} \} / 2A$$

(always want +sqrt)

- Awfully complicated, and sensitive to numerical roundoff
- There must be a simpler scheme (there is ...)

# Conventional Sampling in Bin k

---

```
double linear_sample_std( int *n,
                        double x[],
                        double pdf[],
                        int *k ){
    double x0,p0, x1,p1, r,s, a,b,c,d;

    p0=pdf[*k-1], p1=pdf[*k], x0=x[*k-1], x1=x[*k];
    r = 2.0/((p0+p1)*(x1-x0));
    p0*=r; p1*=r;

    s = (p1-p0)/(x1-x0);
    a = 0.5*s;
    b = p0-s*x0;
    c = .5*s*x0*x0 - p0*x0 - rang();
    d = b*b - 4.*a*c;
    d = (d<0.0) ? 0.0 : d; // sloppy, set negative roundoff to zero

    return .5*(-b+sqrt(d))/a;
}
```

# Conventional Sampling

---

**!=====> in problem setup, given  $x(N)$  &  $pdf(N)$ , find  $cdf(N)$**

```
cdf(1) = 0.0
do k=2,N
  cdf(k) = cdf(k-1) + 0.5*(pdf(k)+pdf(k-1)) * (x(k)-x(k-1))
endo
```

**!=====> during particle histories**

```
!----> random sample bin k
r = rang()
k = binary_search( N, cdf, r )

!----> sample xsample within bin k
xsample = linear_sample( N, x, pdf, k )
```

---

# Optimal Approach to Random Sampling from a Piecewise Linear PDF

# Outline

---

- **Selecting the bin**
  - First try to hand-optimize the search coding
  - Then look at a better algorithm – eliminate the search by alias method
- **Sampling within the bin**
  - Examine an often-used composition method
  - Examine a better composition method
- **Final results**
  - Robust within bin sampling – immune to roundoff & faster
  - No table search, due to alias method
  - Constant time
    - Using linear table search,  $t \sim O(N)$
    - Using binary table search,  $t \sim O(\log N)$
    - Using alias method,  $t \sim O(1)$
  - Overall speedup  $\sim 10\text{-}100\text{x}$  or more

# Binary Search - Variations

## Basic MCNP binary search

```

int binary_search( int*    n,
                  double* table,
                  double* target )
{
    int    jfirst, jlast, jmid;
    jfirst = 0;
    jlast  = *n - 1;

    for(;;) {
        if( jlast-jfirst == 1 ) break;
        jmid = (jfirst+jlast)/2;
        if( *target >= table[jmid] ) {
            jfirst = jmid;
        }
        else {
            jlast  = jmid;
        }
    }
    return  jfirst+1;
}

```

**n=16**            **6.7 ns**  
**n=128**          **14.6 ns**  
**n=1024**         **24.9 ns**

MacBook Pro  
 3.5 GHz I7  
 2.1 Ghz LPDDR3

## Basic binary search, with shift

```

int binary_search1( int*    n,
                   double* table,
                   double* target )
{
    int    jfirst, jlast, jmid;
    jfirst = 0;
    jlast  = *n - 1;

    for(;;) {
        if( jlast-jfirst == 1 ) break;
        jmid = (jfirst+jlast) >> 1;
        if( *target >= table[jmid] ) {
            jfirst = jmid;
        }
        else {
            jlast  = jmid;
        }
    }
    return  jfirst+1;
}

```

**n=16**            **5.4 ns**  
**n=128**          **12.2 ns**  
**n=1024**         **19.0 ns**

# Binary Search - Variations

## Basic binary search, with shift

```
int binary_search1( int*    n,
                  double* table,
                  double* target )
{
    int    jfirst, jlast, jmid;
    jfirst = 0;
    jlast  = *n - 1;

    for(;;) {
        if( jlast-jfirst == 1 ) break;
        jmid = (jfirst+jlast) >> 1;
        if( *target >= table[jmid] ) {
            jfirst = jmid;
        }
        else {
            jlast = jmid;
        }
    }
    return jfirst+1;
}
```

**n=16**            **5.4 ns**  
**n=128**          **12.2 ns**  
**n=1024**         **19.0 ns**

MacBook Pro  
3.5 GHz I7  
2.1 Ghz LPDDR3

## Basic binary search, with shift+merge

```
int binary_search2( int*    n,
                  double* table,
                  double* target )
{
    int    jfirst, jlast, jmid;
    jfirst = 0;
    jlast  = *n - 1;

    for(;;) {
        if( jlast-jfirst == 1 ) break;
        jmid = (jfirst+jlast) >> 1;
        jtest = *target >= table[jmid];

        jfirst = (jtest) ? jmid : jfirst;

        jlast = (jtest) ? jlast : jmid;
    }
    return jfirst+1;
}
```

**n=16**            **5.3 ns**  
**n=128**          **12.4 ns**  
**n=1024**         **19.1 ns**

# Binary Search - Variations

## Basic binary search, with shift+merge

```
int  binary_search2( int*    n,
                    double* table,
                    double* target )
{
    int    jfirst, jlast, jmid;
    jfirst = 0;
    jlast  = *n - 1;

    for(;;) {
        if( jlast-jfirst == 1 ) break;
        jmid  = (jfirst+jlast) >> 1;
        jtest = *target >= table[jmid];
        jfirst = (jtest) ? jmid  : jfirst;
        jlast  = (jtest) ? jlast : jmid;
    }
    return  jfirst+1;
}
```

**n=16**      **5.3 ns**

**n=128**    **12.4 ns**

**n=1024**   **19.1 ns**

## Basic binary search, with goto

```
int  binary_search3( int*    n,
                    double* table,
                    double* target )
{
    int    jfirst, jlast, jmid;
    jfirst = 0;
    jlast  = *n - 1;

    more:
        if( jlast-jfirst == 1 ) goto done;
        jmid  = (jfirst+jlast) >> 1;
        jtest = *target >= table[jmid];
        jfirst = (jtest) ? jmid  : jfirst;
        jlast  = (jtest) ? jlast : jmid;
        goto more;
    done:
    return  jfirst+1;
}
```

**n=16**      **5.6 ns**

**n=128**    **12.2 ns**

**n=1024**   **19.1 ns**

# Binary Search - Variations

## Basic binary search, with shift+merge

```
int  binary_search2( int*    n,
                    double* table,
                    double* target )
{
  int    jfirst, jlast, jmid;
  jfirst = 0;
  jlast  = *n - 1;

  for(;;) {
    if( jlast-jfirst == 1 ) break;
    jmid  = (jfirst+jlast) >> 1;
    jtest = *target >= table[jmid];
    jfirst = (jtest) ? jmid : jfirst;
    jlast  = (jtest) ? jlast : jmid;
  }
  return  jfirst+1;
}
```

**n=16**      **5.3 ns**

**n=128**    **12.4 ns**

**n=1024**   **19.1 ns**

## Basic binary search, with **no if-tests**

```
int  binary_search4( int*    n,
                    double* table,
                    double* target )
{
  int    jfirst, jlast, jmid, k, m;
  jfirst = 0;
  jlast  = *n - 1;
  k      = jlast - jfirst + 1;
  m      = 32 - leadz( &k );
  for( k=0; k<m; k++ ) {
    jmid  = (jfirst+jlast) >> 1;
    jtest = *target >= table[jmid];
    jfirst = (jtest) ? jmid : jfirst;
    jlast  = (jtest) ? jlast : jmid;
  }
  return  jfirst+1;
}
```

**n=16**      **6.6 ns**

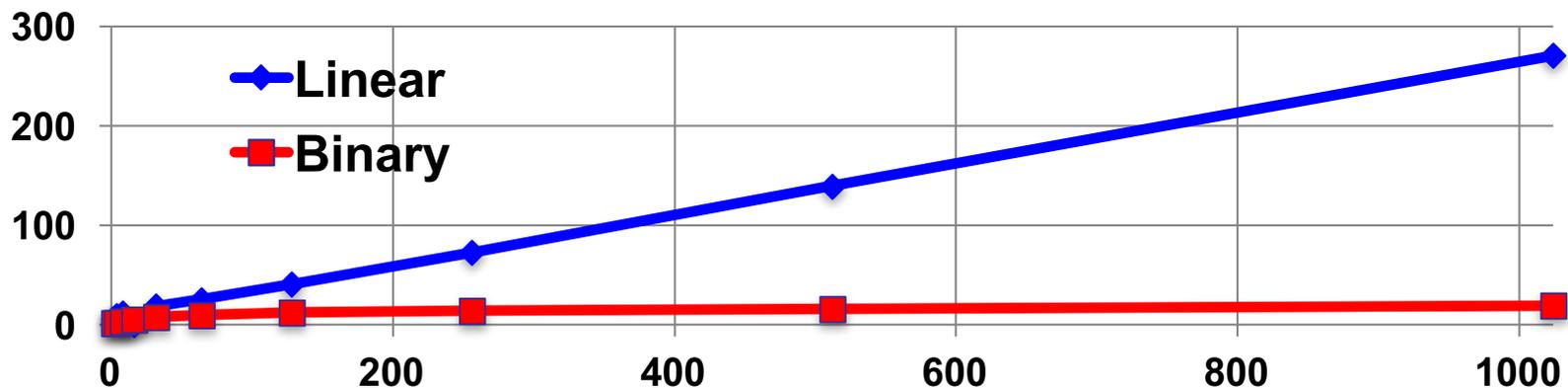
**n=128**    **11.7 ns**

**n=1024**   **17.9 ns**

# Table Search – Timing Summary

- For randomly generated PDFs & randomly sampled targets
- Lookup time (nanosec) vs number of bins:

N bins =	2	4	8	16	32	64	128	256	512	1024
Linear	6.0	9.2	11.7	13.4	18.2	25.2	40.4	72.4	139.4	270.7
Binary	1.6	2.6	4.3	6.7	9.3	11.6	14.6	18.0	21.5	24.9
Binary, shift	1.4	2.2	3.5	5.4	7.3	9.2	12.2	14.1	15.7	19.0
Binary, shift, merge	1.4	2.2	3.5	5.3	7.6	9.4	12.4	14.4	16.2	19.1
Binary, goto	1.3	2.2	3.4	5.6	7.3	9.2	12.2	14.1	15.7	19.1
Binary, no if-tests	2.6	3.6	5.0	6.6	8.2	9.8	11.7	13.8	15.7	17.9



MacBook Pro  
3.5 GHz I7  
2.1 Ghz LPDDR3

# Alias Sampling

## Sampling from Discrete PDF's — Alias Method

Any discrete PDF can be converted into "Alias sampling" form

original PDF:  $\{ f_k \}, \quad k=1, \dots, N$

where  $f_k = \text{probability of selecting } x = x_k$

aliased PDF:  $\{ q_k, i_k \}, \quad k=1, \dots, N$

where  $\frac{1}{N} \cdot q_k = \text{prob. of selecting } \hat{x} = x_k$   
 $\frac{1}{N} \cdot (1 - q_k) = \text{prob. of selecting } \hat{x} = x_{i_k}$

### Alias sampling procedure:

Select uniformly for  $\hat{k}$ :  $\hat{k} \leftarrow \lfloor 1 + N\xi_1 \rfloor$

Select either  $\hat{k}$  or its "alias"  $i_{\hat{k}}$ :  
 if  $\xi_2 < q_{\hat{k}}, \quad \hat{x} \leftarrow x_{\hat{k}},$   
 otherwise,  $\hat{x} \leftarrow x_{i_{\hat{k}}}$

.....(continued on next page)

# Alias Sampling

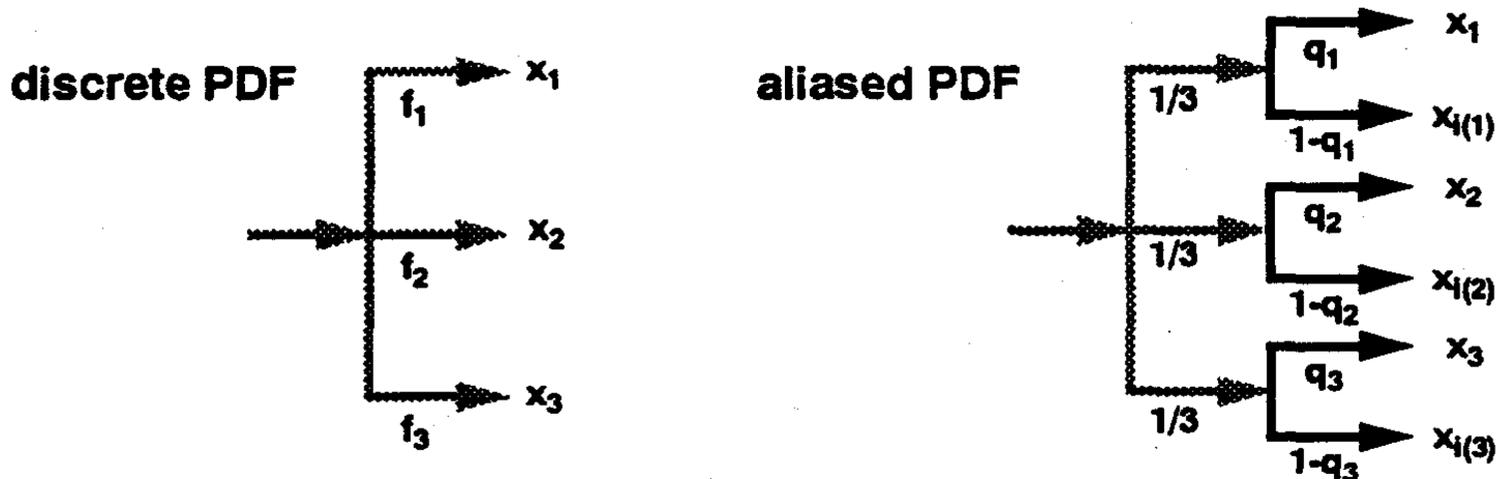
## Sampling from Discrete PDF's — Alias Method (continued)

Why bother with "alias sampling" ?

- No table search needed, requires  $O(1)$  time
- Sampling time is constant & independent of size of PDF
- Vectorizes completely & efficiently
- Fastest possible way to sample discrete PDFs
- Invented by Brown (who later found out Walker did it 3 yr earlier)

Creating the "aliased PDF" amounts to converting an N-way tree from arbitrary branching probabilities with single outcomes to uniform branching probabilities with dual outcomes

(See FB Brown & RACER coding for the set up algorithm)



# Alias Sampling - Setup

---

```

void alias_setup(  int*      n,
                  double* prob,
                  double* aiq  )
{
  // set up aiq[] array for alias sampling,
  // using FB Brown method, with 1-based indexing for aiq()
  double eps = 1e-10, onep=1e0+eps, onem=1e0-eps;
  int     is[*n], ig[*n], j, js, jg, ls, lg;
  double  p[*n];

  // initial index lists of smaller/greater
  ls = -1;
  lg = -1;
  for( j=0; j<*n; j++ ) {
    p[j] = (*n) * prob[j];
    if(     p[j]<onem )  is[++ls] = j;
    else if( p[j]>onep ) ig[++lg] = j;
  }

  // fill the aiq[] array
  for( j=0; j<*n; j++ ) aiq[j] = j;
  lg = 0;
  while( ls>=0 ) {
    js = is[ls--];
    jg = ig[lg];
    aiq[js] = jg + p[js];      // aiq = (index of alias).(prob of non-alias)
    p[jg] += p[js] - 1e0;
    if( p[jg]<onem ) is[++ls] = ig[lg];
    if( p[jg]<onep ) lg++;
  }
  // change from 0-based to 1-based for aiq[]
  for( j=0; j<*n; j++ ) aiq[j] += 1e0;
  return;
}

```

# Alias Sampling - Sample

---

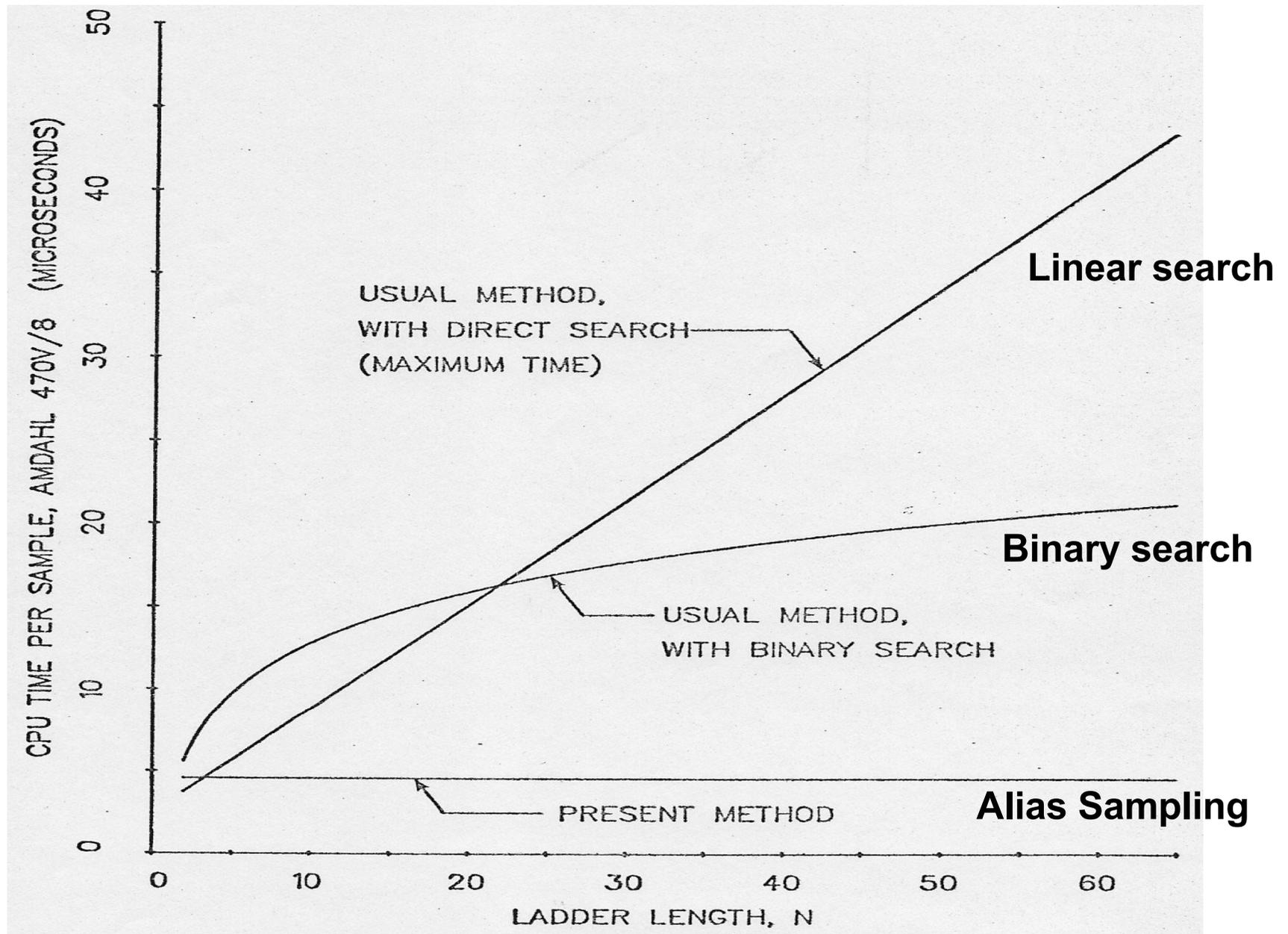
```
int alias_sample( int*    n,
                  double* aiq )
{
    // use alias sampling,
    // return index in range [1,n]
    int    bin, alias;
    double  r,    q;

    r = (*n)*rang();    bin    = r;    r -= bin;
    q = aiq[bin++];    alias = q;    q -= alias;

    if( r>q )    bin = alias;

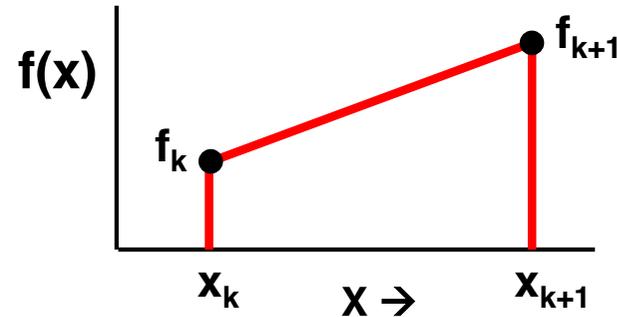
    return  bin;
}
```

# Alias Sampling - Timing



# Continuous PDFs – Linear

## Example - Sampling from linear PDF bin k



Note: normalized such that  $0.5 \cdot (f_k + f_{k+1}) \cdot (x_{k+1} - x_k) = 1$

**PDF:**  $f(x) = f_k + m(x - x_k), \quad m = (f_{k+1} - f_k) / (x_{k+1} - x_k), \quad x_k \leq x \leq x_{k+1}$

**CDF:** 
$$F(x) = (m/2)x^2 + (f_k - mx_k)x + (mx_k^2/2 - f_k x_k)$$

$$= A x^2 + B x + C$$

**Sampling scheme:**  $F(x) = \xi, \quad \text{solve for } x$   
 $x = \{ -B + \sqrt{B^2 - 4A[C - \xi]} \} / 2A$

(always want +sqrt)

- Awfully complicated, and sensitive to numerical roundoff
- There must be a simpler scheme (there is ...)

# Continuous PDFs – Linear

## Example - Sampling from linear PDF in range $[a,b]$ , 1 bin

### Composition method #1

Decompose the original PDF into the sum of 2 PDFs, uniform + linear:

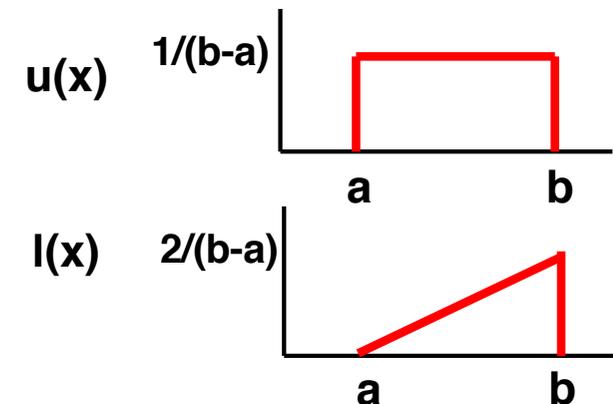
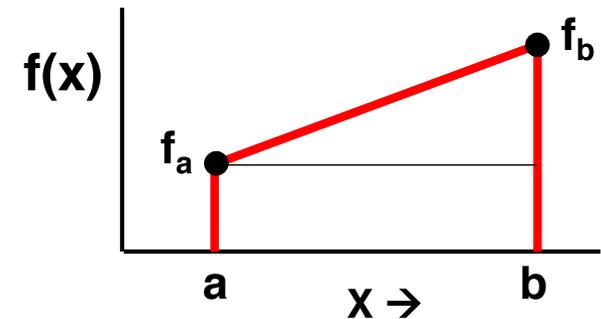
$$f(x) = p_u u(x) + p_l l(x)$$

$u(x)$  = uniform on  $a \leq x \leq b$ ,

$$p_u = \{ \min(f_a, f_b) (b-a) \} / \{ .5(f_a + f_b) (b-a) \}$$

$l(x)$  = linear on  $a \leq x \leq b$ ,

$$p_l = \{ .5 \text{abs}(f_b - f_a) (b-a) \} / \{ .5(f_a + f_b) (b-a) \}$$



Sampling scheme: if(  $\xi_1 < p_u$  )

$$x \leftarrow a + (b-a) \xi_2$$

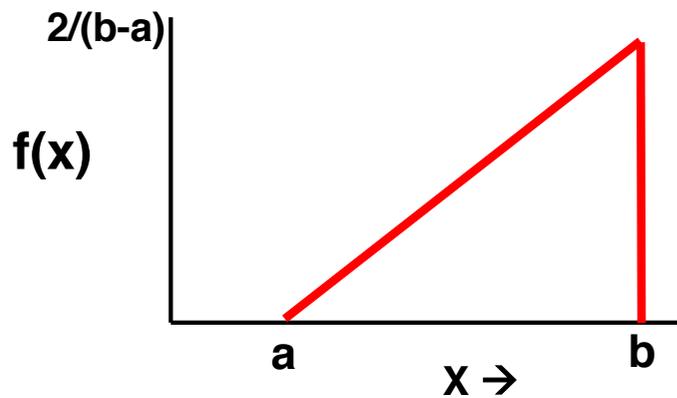
else

$$\text{if}( f_b > f_a ) \quad x \leftarrow a + (b-a) \text{sqrt}( \xi_2 )$$

$$\text{else} \quad x \leftarrow a + (b-a) (1 - \text{sqrt}( \xi_2 ))$$

# Continuous PDFs – Linear

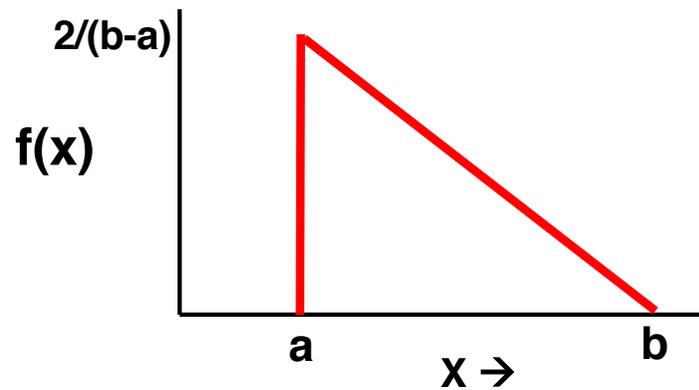
## Increasing linear PDF



Random sampling can be done with a simple shifting & scaling of the unit PDF:

$$x \leftarrow a + (b-a) \sqrt{\xi}$$

## Decreasing linear PDF



Random sampling can be done with a simple shifting & scaling of the unit PDF:

$$x \leftarrow a + (b-a) (1 - \sqrt{\xi})$$

# Continuous PDFs – Linear

## Example - Sampling from linear PDF in range $[a,b]$ , 1 bin

### Composition method #2

Decompose the original PDF into the sum of 2 PDFs, increasing + decreasing linear:

$$f(x) = p_m m(x) + p_l l(x)$$

$m(x)$  = linear decreasing on  $a \leq x \leq b$ ,

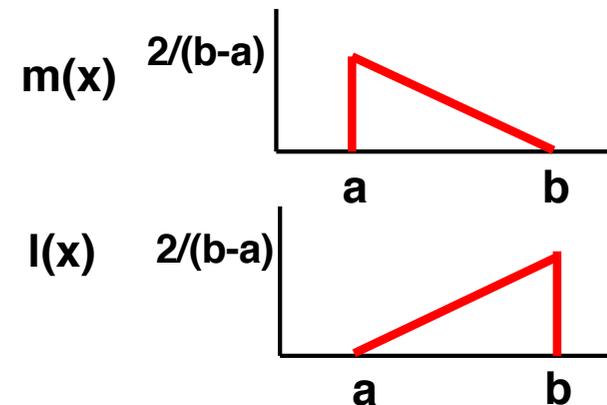
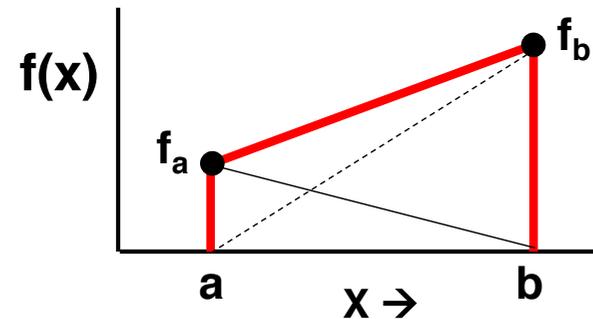
$$p_m = \{ .5 f_a (b-a) \} / \{ .5(f_a+f_b) (b-a) \}$$

$$= f_a / (f_a+f_b)$$

$l(x)$  = linear increasing on  $a \leq x \leq b$ ,

$$p_l = \{ .5 f_b (b-a) \} / \{ .5(f_a+f_b) (b-a) \}$$

$$= f_b / (f_a+f_b)$$



**Sampling scheme:**

if(  $\xi_1 < p_l$  )

$x \leftarrow a + (b-a) \text{sqrt}(\xi_2)$

else

$x \leftarrow a + (b-a) (1 - \text{sqrt}(\xi_2))$

# Composition Method for Sampling within Bin $k$

---

```
double linear_sample_new( int *n, // num of points
                          double x[], // x[*n]
                          double pdf[], // pdf[*n]
                          int *k ) // bin number,
                                  // 1-based
{
    double r, p;

    // next line could have been precomputed, in place of pdf[]
    p = pdf[*k] / (pdf[*k-1] + pdf[*k]);

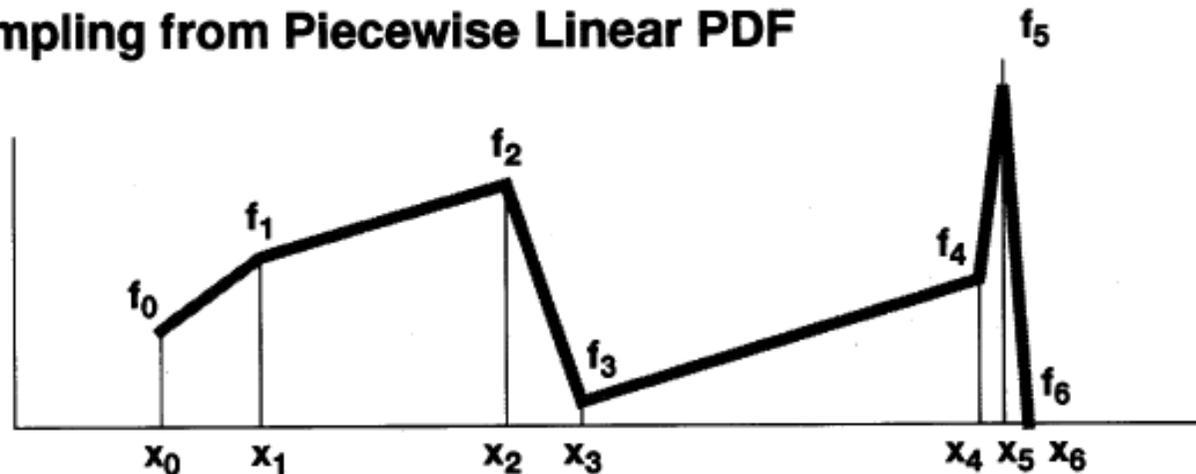
    r = sqrt( rang() );

    if( rang() > p ) r = 1.0-r;

    return x[*k-1] + (x[*k]-x[*k-1])*r;
}
```

# Continuous PDFs – Linear

## Examples — Sampling from Piecewise Linear PDF



**Two-step sampling:**

- (1) Sample from discrete PDF to select a bin
- (2) Sample from linear PDF within bin

• Discrete PDF: 
$$p_k = \frac{(f_k + f_{k-1})}{2} \cdot (x_k - x_{k-1}), \quad k = 1, \dots, N$$

- generate  $\xi$
- use table search or alias method to select  $\mathbf{K}$

• Linear sampling within bin  $\mathbf{K}$ :

- generate  $\xi$
- then,
 

if $\xi_1 < \frac{f_{k-1}}{f_k + f_{k-1}},$	$\hat{x} \leftarrow x_k - (x_k - x_{k-1}) \sqrt{\xi_2}$
otherwise	$\hat{x} \leftarrow x_{k-1} + (x_k - x_{k-1}) \sqrt{\xi_2}$

# Combined Alias & Linear PDF Sampling

---

- **Combined:** alias sampling to select a bin, then composition method for linear pdf sampling

```
double alias_sample_linear_pdf( int *npts,
                                double x[], // [npts]
                                double pdf[], // [npts]
                                double aiq[] ) // [npts-1]
{
    // Note: below uses 0-based indexing, C-style
    int nbin, bin, alias;
    double r, q;

    nbin = *npts - 1;

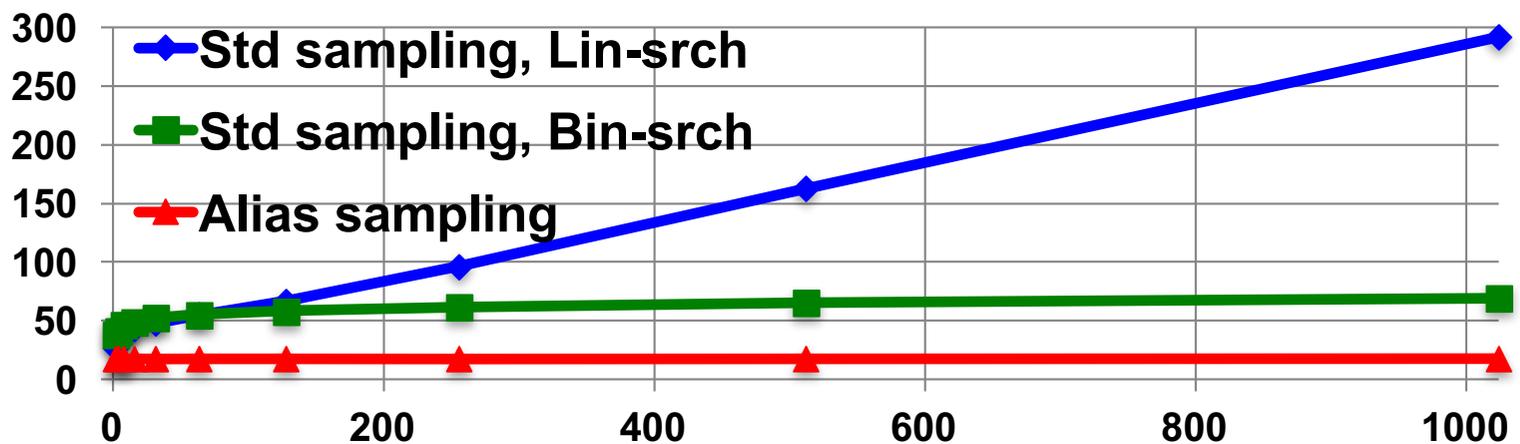
    // use alias sampling, get bin in range [0,nbin-1]
    r = nbin*rang(); bin = r; r -= bin;
    q = aiq[bin]; alias = q; q -= alias;
    if( r>q ) bin = --alias;

    // linear sampling within bin, composition method
    r = sqrt( rang() );
    if( rang()*(pdf[bin]+pdf[bin+1]) > pdf[bin+1] ) r = 1.0-r;
    return x[bin] + (x[bin+1]-x[bin])*r;
}
```

# Comparison of Sampling Times

- **Compare:**
  - Standard sampling, with linear search
  - Standard sampling, with binary search
  - Alias sampling, with composition method
- **Sampling time (nanosec) vs Number of bins**

N bins =	2	4	8	16	32	64	128	256	512	1024
Std sampling, linear search	26.6	34.7	41.7	43.4	47.6	54.6	66.5	96.1	162.6	291.8
Std sampling, binary search	37.9	41.4	46	48.7	52.1	55.1	58.1	61.7	65	69.1
Alias sampling	20.5	20.7	21.1	20.7	20.5	20.6	20.5	20.4	20.6	20.7
Alias, combined	17.6	17.4	17.5	17.4	17.3	17.4	17.4	17.3	17.4	17.6



MacBook Pro  
3.5 GHz I7  
2.1 Ghz LPDDR3

# Summary

---

- **Optimal sampling method for piecewise-linear PDF**
  - Alias sampling to select bin (eliminates the search)
  - Composition method using increasing/decreasing linear sampling
- **Final results**
  - Robust within bin sampling – immune to roundoff & faster
  - No difficulties if x-points are identical
  - Can mix delta functions into piecewise-linear PDF
  - No table search, due to alias method
  - Constant time
    - Using linear table search,  $t \sim O(N)$
    - Using binary table search,  $t \sim O(\log N)$
    - Using alias method,  $t \sim O(1)$
  - Overall speedup  $\sim 10\text{-}100\text{x}$  or more

## References - Sampling

---

- F.B. Brown, "Monte Carlo Techniques for Nuclear Systems – Theory Lectures", Lecture c-t-02 Random Sampling, LA-UR-16-29043 (2016)
- D. E. Knuth, The Art of Computer Programming, Vol. 2: Semi-numerical Algorithms, 3<sup>rd</sup> Edition, Addison-Wesley, Reading, MA (1998).
- D.E. Knuth, The Art of Computer Programming Vol 3 - Sorting & Searching, 3<sup>rd</sup> Edition, Addison-Wesley, Reading, MA (1998).
- L. Devroye, Non-Uniform Random Variate Generation, Springer-Verlag, NY (1986).
- J. von Neumann, "Various Techniques Used in Conjunction with Random Digits," *J. Res. Nat. Bur. Stand. Appl. Math Series* 3, 36-38 (1951).
- C. J. Everett and E. D. Cashwell, "A Third Monte Carlo Sampler," LA9721-MS, Los Alamos National Laboratory, Los Alamos, NM (1983).
- H. Kahn, "Applications of Monte Carlo," AECU-3259, Rand Corporation, Santa Monica, CA (1954).

## References – Alias Sampling

---

- A.J. Walker, “An Efficient Method for Generating Discrete Random Variables with Generalized Distributions”, ACM Trans. Math. Software, Vol 3, No. 3, 253-256 (Sept, 1977)**
- R.A. Kronmal, A.V. Peterson Jr, “On the Alias Method for Generating Random Variables from a Discrete Distribution”, The American Statistician, Vol 33, No 4, 214-218 (Nov 1979)**
- F.B. Brown, W.R. Martin, D.A. Callahan, “A Discrete Sampling Method for Vectorized Monte Carlo Calculations”, Trans. Am. Nucl. Soc. 38, 354 (1981)**
- M.D. Vose, “A Linear Algorithm for Generating Random Numbers with a Given Distribution”, IEEE Trans. Software Engineering, Vol 17, No. 9 (Sept, 1991)**
- G. Marsaglia, "Generating Discrete Random Variables in a Computer," Comm. Assoc. of Computing Machinery, 6, 37-38 (1963)**





**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Permutations, Sets of N-from-M, & Counting-Sorts**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Outline

---

- **Random Permutations**
- **Sampling N Items from a Set of M Items**
  - **With Replacement**
  - **Without Replacement**
- **Reordering the Fission Bank, without Sorting**

---

# Random Permutations

# Random Permutations (1)

---

- **Problem:** Generate a random permutation of a set of N items
  - N items:  $\{ x_1, x_2, x_3, \dots, x_N \}$
  - Want a a random ordering of the N items, without duplicate or missing entries
  - Examples: shuffling cards; random order for presentations; . . .
- **Basic algorithm**

for  $J = 1 .. N$

    pick a random integer K in range [1,N]

    swap  $x(J)$  and  $x(K)$

# Random Permutations (2)

---

## Matlab

```
for J = 1 : N

    % Random integer in range [1..N]
    K = 1 + floor( N*rand );

    % Swap x(J) & x(K)
    xtmp = x(J);
    x(J) = x(K);
    x(K) = xtmp;

end
```

## C

```
long    J, K, N;
double  x[], xtmp;

for( J=0; J<N; J++ ) {

    K =floor( N*rang() );

    xtmp = x[J];
    x[J] = x[K];
    x[K] = xtmp;

}
```

---

# Sampling N-from-M Items

# Sampling N from M Items (1)

---

- **Problem:** Given M items, randomly select N
- **For  $N \leq M$** 
  - If duplicates are allowed, called "sampling with replacement"
  - If duplicates are not allowed, called "sampling without replacement"
- **For  $N > M$** 
  - Usually interpreted to mean:
    - (1) Make K copies of all M items, where  $K = \text{floor}(N / M)$
    - (2) Sample the remainder  $(N - K * M)$  with or without replacement

**Example:**

To sample 20 items from 6:

Copy all 6 items 3 times each,  
then sample 2 items from 6

- While we may be picking from  $\{x_1, x_2, x_3, \dots, x_N\}$ , we only need consider the indices of selected items. After picking the list of indices, gather the values.

## Sampling N from M Items (2)

---

- **Sampling "with" vs "without" replacement**
  - **Easy way to understand - picking Powerball numbers, by drawing labeled balls from bucket (pick 5 from 69, then 1 from 26)**
    - Sampling **with** replacement:  
pick a ball, record the number, then put it back in the bucket
    - Sampling **without** replacement:  
pick a ball, record the number, **don't** put it back in the bucket
    - Sampling **with** replacement could give: 5, 5, 5, 5, 5, 5
    - Sampling **without** replacement gives 5 unique numbers, then another.
  - **Need sampling without replacement for picking Powerball numbers**
- **This type of sampling occurs in criticality calculations, where N neutrons must be selected randomly from a fission-neutron-bank that contains M neutrons**
  - **We generally prefer to use sampling without replacement**

# Sampling N from M Items, WITH REPLACEMENT (3)

---

- Example

Given:  $M=5, \{1, 2, 3, 4, 5\}$

Randomly select:  $N=3$  items, with replacement

```
integer, parameter :: M=5           ! Given items
integer, parameter :: N=3           ! How many to select

do J = 1, N

    K = 1 + M*rang()                 ! Random pick from 1..M

    keep(J) = K                       ! Save the pick

enddo
```

# Sampling N from M Items (4)

---

- Example

Given:  $M=5, \{1, 2, 3, 4, 5\}$

Randomly select:  $N=3$  items, without replacement

```
integer, parameter :: M=5      ! Given items
integer, parameter :: N=3      ! How many to select

IX(1:M) = [ 1, 2, 3, 4, 5 ]    ! List of items

Mleft = M
do J = 1, N
  K = 1 + Mleft*rang()          ! Random pick from items left

  keep(J) = K                   ! Save the pick

  IX(K) = IX(Mleft)             ! Replace pick by last item
  Mleft = Mleft - 1             ! Fix count of unpicked items
enddo
```

# Sampling N from M Items (5)

---

- Better algorithm - from Knuth, Volume 2, Section 3.4.2
- Example

Given:  $M=5, \{1, 2, 3, 4, 5\}$   
 Randomly select:  $N=3$  items, without replacement

```
integer, parameter :: M=5           ! Given items
integer, parameter :: N=3           ! How many to select

K = 0                                ! # selected so far
do J = 1, M                           ! Note: M, not N
  prob = real(N-K) / real(M-J+1)      ! Prob of selecting
  if( rang() < prob ) then
    K = K + 1
    keep(K) = J                       ! Save it
    if( K==N ) exit
  endif
enddo
```

# Sampling N from M Items (6)

---

- **Example - randomly pick numbers for Powerball**

Use sampling without replacement to pick 5 from 69

Then pick 1 from 26

[As of 2015]

## Notes:

- Using the first algorithm for sampling without replacement, the results are not ordered, so may get [ 5, 1, 3, 2, 4, 6 ]
- Using the Knuth algorithm, results are ordered, so would get [ 1, 2, 3, 4, 5, 6 ]

# Sampling N from M Items - Mods

---

- Modifications so that algorithm works for  $N < M$ ,  $N = M$ ,  $N > M$
- Example

Given:  $M=5, \{ 1, 2, 3, 4, 5 \}$   
 Randomly select:  $N=3$  items, without replacement

```
integer, parameter :: M=5      ! Given items
integer, parameter :: N=3      ! How many to select

K = 0                          ! # selected so far
do J = 1, M                    ! Note: M, not N
  prob = real(N-K) / real(M-J+1) ! Prob of selecting
  knt = prob + rang()
  do i=1,knt
    K = K + 1
    keep(K) = J                ! Save it
    if( k==N ) exit
  endif
enddo
```

# Sampling N from M Weighted Items (W/O Replacement)

- When the M items to be sampled (without replacement) each have weights, only minor modifications are needed
- Algorithm below works for  $N < M$ ,  $N = M$ ,  $N > M$
- Example

Given:  $M=5, \{ 1, 2, 3, 4, 5. \}$

$W= \{ w_1, w_2, w_3, w_4, w_5 \}$

Randomly select:  $N=3$  items, without replacement

```
integer, parameter :: M=5           ! Given items
integer, parameter :: N=3           ! How many to select

K      = 0                           ! # selected so far
wtot = sum(W)
wcum = 0                             ! cumulative wgt, so far
do J = 1, M                          ! Note: M, not N
  prob = w(J) * real(N-K) / (wtot-wcum) ! Prob of selecting
  wcum = wcum + w(J)
  knt = prob + rang()
  do i=1,knt
    K      = K + 1
    keep(K) = J                       ! Save it
    if( K==N ) exit
  enddo
enddo
enddo
```

# Fission Bank Reordering

# Fission Bank Reordering (1)

---

- During criticality calculations, neutrons created by fission during a cycle are added to the "fission bank", and held as sources for the next cycle
  - Due to parallel processing (threading &/or MPI), the order of the neutrons in the fission bank is not predictable
  - For reproducible results in criticality problems, the fission bank must be reordered into a unique order prior to starting the next cycle
  - For definiteness, we choose to order the fission bank according to the "particle number" nps. If there are more than 1 fission bank entries with the same nps, retain the order of those.
- Fission bank example – showing just nps, xyz

original			reordered	
3	xyz...		1	xyz...(a)
1	xyz...(a)		1	xyz...(b)
4	xyz...	→	2	xyz...
1	xyz...(b)		3	xyz...
2	xyz		4	xyz...

## Fission Bank Reordering (2)

---

- The fission bank reordering could be done by sorting, but that would take  $O(N^2)$  or  $O(N \log N)$  time, and could be expensive
- A **counting sort** algorithm is most efficient for reordering the fission bank,  $O(N)$  timing
  - Sorts a collection of objects according to keys that are small integers
  - Applies only to sorting integers
  - Basic idea:
    - count the number of objects that have each distinct key value
    - use arithmetic on those counts to determine the positions of each key value in the output sequence
  - Running time is linear in the number of items and the difference between the maximum and minimum key values
  - Suitable for cases where the range of keys is not significantly greater than the number of items

## Fission Bank Reordering (3)

---

- Algorithm described next does the reordering in  $O(N)$  time, & is the method used in the RACER & MCNP codes

FB Brown & TM Sutton, "Reproducibility and Monte Carlo Eigenvalue Calculations", Trans Am Nuc Soc 65, 235 (1992)

Given initial vector of parent numbers in the bank,  $P(N)$

$$L_1 = 1$$

$$L_{J+1} = L_J + \text{count}[P_I == J], \quad J = 1, \dots, N$$

So that  $(L_{J+1} - L_J) = \text{number of progeny in bank for parent } J$

Then permutation vector  $Q(N)$  for reordering  $P(N)$  is

for  $J=1, N$

$$Q_J = L_{P(J)}$$

$$L_{P(J)} = L_{P(J)} + 1$$

# Fission Bank Reordering (4)

---

```

!==> find permutation vector for reordering an array in increasing order
      n          = length of ix() & perm()
      ix(:)      = integer vector, unchanged
      perm(:)    = perm vector for reordering ix()

keymin = minval(ix(1:n))      ! minimum ix()
keymax = maxval(ix(1:n))     ! maximum ix()
nkeys  = keymax - keymin + 1  ! size of vector to span range of ix()
allocate( knt(nkeys) )

knt(1:nkeys) = 0
do i=1,n
  key      = ix(i) - keymin + 1
  knt(key) = knt(key) + 1     ! count the entries for each unique ix()
enddo

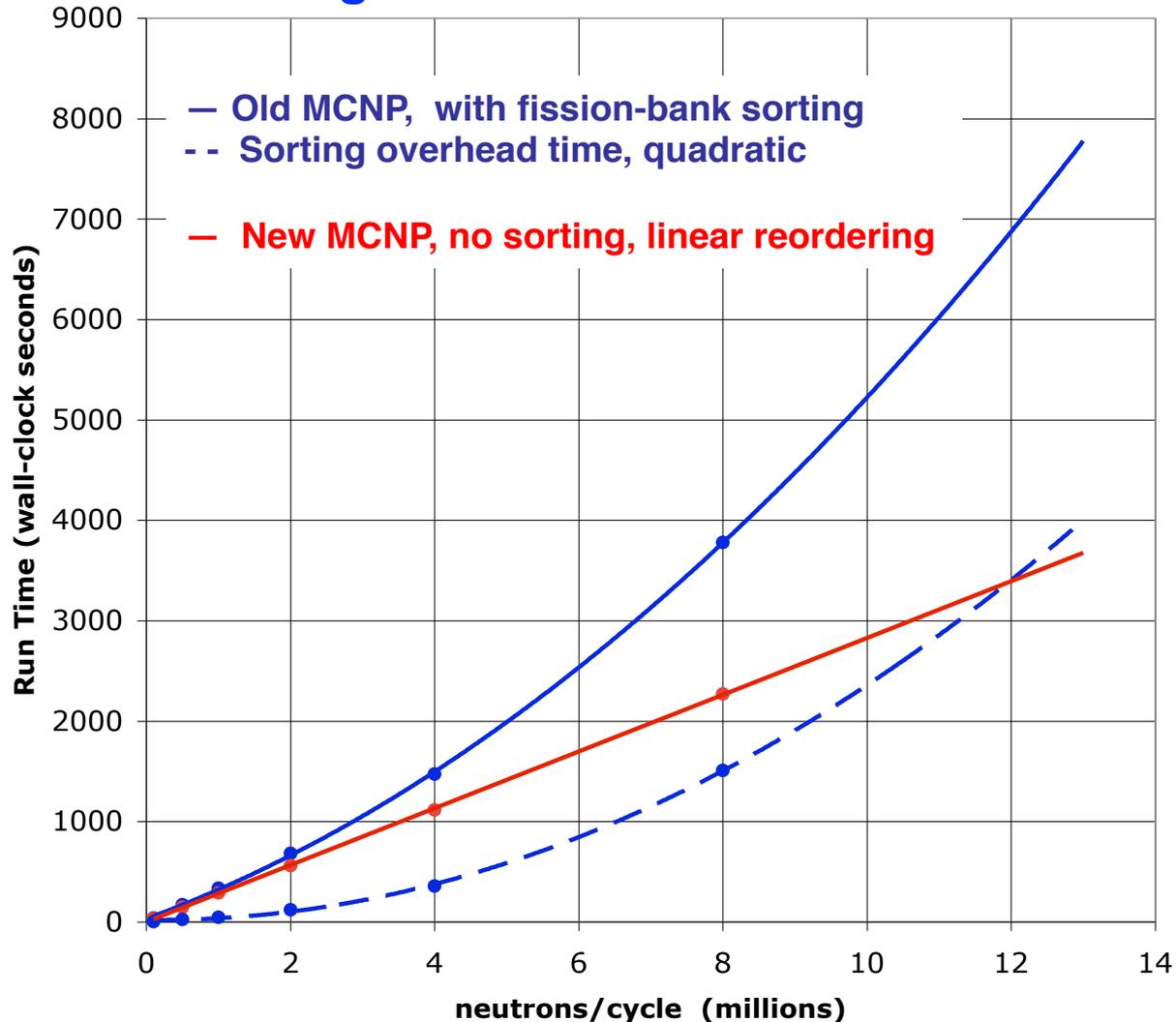
loc = 1
do key=1,nkeys
  km      = knt(key)
  knt(key) = loc              ! convert to starting locs in permuted vect
  loc     = loc + km
enddo

do i=1,n
  key      = ix(i) - keymin + 1
  loc      = knt(key)         ! get loc for the permuted entry
  perm(loc) = i              ! store index for permuted entry
  knt(key) = knt(key) + 1     ! bump the base loc, in case duplicates
enddo
deallocate( knt )

```

# Fission Bank Reordering (5)

## Timing Studies vs Fission Bank Size



## PWR2D Model

- 1/4-core, detailed geometry, ENDF/B-VII
- KCODE problem, first 5 cycles
- Mac Pro, 3 GHz, 2 quad-core Xeon
- Run with 8 threads
- Times are wall-clock seconds
- Identical results for old & new reordering

# Fission Bank Reordering (6)

---

- **For older versions of mcnp**
  - Fission bank reordering was done using a simple-minded bubble-sort, that scaled as  $O(N^2)$
  - Timing was OK for 100s or 1000s of neutrons/cycle
  - For millions of neutrons/cycle, the time for reordering was longer than it took to run the neutron histories !
  
- **For newer versions of mcnp (and racer)**
  - Counting-sort, timing is  $O(N)$
  - Time for reordering fission bank is not an issue

Note: Romano's papers compared his methods for treating the fission bank with the older mcnp schemes. Not valid for newer schemes.







**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Some Ideas for a New Random Number Generator**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Outline

---

- **MCNP RNG History**
- **RNG Period & Modern Calculations**
- **RN Usage & Reproducibility**
- **Reminders**
- **Detailed Review of RNG #2**
- **MCGs & Different Adders**
- **RNG #2 Correlation Study**
- **Summary**

# MCNP RNG History

LCG = linear congruential generator  
MCG = mixed congruential generator

- Prior to 2002, MCNP used a 48-bit RNG

48-bit LCG, period =  $2^{46} \sim 7.04 \times 10^{13}$

$$s_{k+1} = g \cdot s_k \text{ mod } 2^{48}, \quad \text{with } g = 5^{19}, s_0 = \text{odd}$$

- In 2002, MCNP5 introduced the current 64-bit RNG

48-bit LCG, period =  $2^{46} \sim 7.04 \times 10^{13}$  (default)

$$s_{k+1} = g \cdot s_k \text{ mod } 2^{48}, \quad \text{with } g = 5^{19}, s_0 = \text{odd}$$

63-bit LCG, period =  $2^{61} \sim 2.31 \times 10^{18}$  (32k longer than default)

$$s_{k+1} = g \cdot s_k \text{ mod } 2^{63}, \quad \text{with } g = (3 \text{ choices}), s_0 = \text{odd}$$

63-bit MCG, period =  $2^{63} \sim 9.22 \times 10^{18}$  (128k longer than default)

$$s_{k+1} = g \cdot s_k + c \text{ mod } 2^{63}, \quad \text{with } g = (3 \text{ choices}), c = 1$$

- For all, default stride = 152,917

# RNG Period & Modern Calculations

---

- **With the default stride (152917), how many histories can be run without exceeding the RNG period?**
  - 48-bit LCG:  $\sim 4.60 \times 10^8$
  - 63-bit LCG:  $\sim 1.51 \times 10^{13}$
  - 63-bit MCG:  $\sim 6.03 \times 10^{13}$
- **For many applications, people are now routinely running  $10^9$ - $10^{10}$  histories, using threading &/or MPI**
  - **For a calculation with  $10^{10}$  histories:**
    - using 48-bit LCG: cycle through entire RN sequence 21.7 times
    - using 63-bit LCG: use only 6.6 % of RN sequence
    - using 63-bit MCG: use only 1.7 % of RN sequence
  - **It's time to consider changing the RNG default to extend the period, and possibly to extend the stride as well**

# RNG Usage & Reproducibility

---

- **A single RNG is used for all events within a history**
  - Using **mode n p** gives different neutron results than using **mode n**
  - Adding a tally that involves cross-sections changes other results
  - Adding dxtran or a point-detector changes other results
  - Other examples .....
- **We could restore reproducibility by using different RNGs for different classes of events**
  - RNG-A for neutron tracking
  - RNG-B for cross-sections (eg, URR)
  - RNG-C for photons
  - RNG-D for dxtran
  - RNG-E for point-detectors
  - etc.
  - **Where would we get all of these independent RNGs ?**

# Reminders

---

- **The RNG for MCNP has a number of requirements**
  - Small state size
  - Solid number-theory basis
  - Fast
  - Fast skip-ahead algorithm (required for parallel)
  - Robust
  - Thoroughly tested (by us)
- **At present, there are no reasons to abandon the MCG approach**
  - Would require more than 6 months for R&D and testing
  - Current MCG can handle  $6 \times 10^{13}$  histories without repetition
- **The current MCG is readily extensible to provide different RN sequences for different classes of events**

# Detailed Review of MCNP RNG #2

---

- **MCNP RNG #2**

$$s_{k+1} = g \cdot s_k + c \text{ mod } 2^{63}$$

**g = multiplier = 9219741426499971445**

**c = adder = 1**

**integer bits = 63**

**modulus =  $2^{63}$**

**period =  $2^{63} = 9.2234E+18$**

**stride = 152917**

**largest RN = 9.9999999999999998890E-01**

**smallest RN = 1.110223024625156540E-16**

- **Knuth checks on multiplier, modulus, & adder – OK**
  - For MCG:**
    - (1) c is coprime with modulus**
    - (2) (g-1) is multiple of every prime factor of modulus**
    - (3) (g-1) is multiple of 4 if modulus is multiple of 4**
- **Excellent results on spectral test (hyperplane spacing)**
- **Passes all statistical tests in DIEHARD, L'Ecuyer, Vattulainen, & Mascagni test suites**

# Detailed Review of MCNP RNG #2

- L'Ecuyer asserted that RNG #2 had excellent results for the spectral test
- My testing results:

Knuth criteria, for  $2 \leq t \leq 6$

$\mu_t > 1$             excellent, pass

$0.1 \leq \mu_t \leq 1$     pass

$\mu_t \leq 1$             fail

For  $S_t$ , closer to 1.0 is better

```

RN Generator 2                g=9219741426499971445, c=1, m=63
t=2      mu(t)=2.85089031774985996530      S1(t)=.88650386627857186441
t=3      mu(t)=2.80455905390264406911      S1(t)=.77938790282751717292
t=4      mu(t)=3.57264878020696490964      S1(t)=.77566223017859449159
t=5      mu(t)=3.83795429041141295503      S1(t)=.76252004748034724096
t=6      mu(t)=3.82954881918753935792      S1(t)=.73714566150742348291
t=7      mu(t)=6.42411095559202805217      S1(t)=.77633463775048459411
t=8      mu(t)=6.81135485905833985083      S1(t)=.75438085713451985058

***** PASSED - EXCELLENT *****

```

For comparison, the mcnp default RNG gives these spectral test results:

```

RN Generator 1                g=19073486328125, c=0, m=48
t=2      mu(t)=3.02330351881511771611      S1(t)=.91291691559340599922
t=3      mu(t)=.19696317112830573650      S1(t)=.32155885499949565800
t=4      mu(t)=1.88704849748587688769      S1(t)=.66125783229413364207
t=5      mu(t)=.94833720772801102363      S1(t)=.57653259365188430775
t=6      mu(t)=1.85974307624174089346      S1(t)=.65353803883396297387
t=7      mu(t)=.88018197696895914615      S1(t)=.58442513153953261525
t=8      mu(t)=1.29313543514425238928      S1(t)=.61290331823653343011

***** PASSED *****

```

# MCGs & Different Adders

- **MCG:**  $s_{k+1} = g \cdot s_k + c \pmod{2^{63}}$
- **If Knuth's criteria for  $g$ ,  $c$ , & modulus hold, the statistical properties of an MCG do not depend on the adder  $c$** 
  - Spectral test is independent of the adder  $c$
  - Using MCGs with the same multiplier & modulus, but different adders, will produce different RN sequences. They all have the identical statistical properties & pass the same statistical tests
- **Regarding MCNP RNG #2:**
  - All of these provide independent RN sequences, with identical statistical properties:

$$s_{k+1} = g \cdot s_k + 1 \pmod{2^{63}} \quad \text{excellent}$$

$$s_{k+1} = g \cdot s_k + 3 \pmod{2^{63}} \quad \text{excellent}$$

$$s_{k+1} = g \cdot s_k + 5 \pmod{2^{63}} \quad \text{excellent}$$

$$s_{k+1} = g \cdot s_k + 7 \pmod{2^{63}} \quad \text{excellent}$$

.....

Adder  $c$  must be chosen such that  $\text{GCD}(c, 2^{63}) = 1$ ,  
 $c$  can be any odd number

## RNG #2 – Correlation Study

---

- **Using RNG #2 with different adders**

- Same statistical properties

- Is there any correlation in the RN sequences if different adders are used? **NO**

- **Test results on next pages**

For adders = [ 1, 3, 5, 7, 9, 11, 13, 15, 17 ]

For 10,000 particles & stride 152917

Generate all RNs (within stride) for a particle & adder

Then,

Compute correlation coef's for lags [0,...,8] for given particle & adder

Compute cross-correlation for lags [0,...,8] between different adders

Average the correlations & cross-correlations over all particles

Also get min & max correlations & cross-correlations for single particle

13.76 billion RNs in all, easy to do more



# RNG #2 – Correlation Study

---

```

-----> correlation coefficients, AVERAGE for nps = 1, ..., 10000
RNG lag= 0      1      2      3      4      5      6      7      8
  0  1.0000 -0.0000 -0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000 -0.0000
  1  1.0000 -0.0000  0.0000  0.0000 -0.0000 -0.0001  0.0000 -0.0001 -0.0000
  2  1.0000 -0.0000  0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000 -0.0000
  3  1.0000 -0.0000 -0.0000  0.0000 -0.0000  0.0000 -0.0001 -0.0000 -0.0000
  4  1.0000  0.0000  0.0000  0.0000 -0.0000 -0.0001 -0.0000 -0.0000  0.0000
  5  1.0000 -0.0000  0.0000  0.0000 -0.0000 -0.0000  0.0000  0.0000 -0.0000
  6  1.0000 -0.0000 -0.0001 -0.0000  0.0000 -0.0000 -0.0001 -0.0000  0.0000
  7  1.0000  0.0001 -0.0000  0.0000 -0.0000  0.0000  0.0000  0.0000  0.0000
  8  1.0000 -0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000

```

```

-----> cross-correlation, AVERAGE for nps = 1, ..., 10000
RNG lag= 0      1      2      3      4      5      6      7      8
0-0  1.0000 -0.0000 -0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000 -0.0000
0-1  0.0000  0.0000 -0.0001  0.0000  0.0000  0.0000  0.0000 -0.0000 -0.0000
0-2 -0.0000 -0.0000  0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000  0.0000
0-3  0.0000 -0.0000 -0.0000 -0.0000 -0.0001 -0.0000  0.0000 -0.0000 -0.0000
0-4 -0.0000 -0.0000  0.0000  0.0000 -0.0000  0.0000 -0.0000 -0.0000 -0.0000
0-5 -0.0001  0.0001 -0.0000  0.0000 -0.0000 -0.0000  0.0000 -0.0000 -0.0000
0-6  0.0000  0.0000  0.0000  0.0000 -0.0000  0.0000 -0.0000  0.0000 -0.0000
0-7 -0.0000 -0.0001 -0.0000  0.0001 -0.0000  0.0001 -0.0000 -0.0000  0.0000
0-8 -0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000 -0.0000  0.0000 -0.0000

```

# RNG #2 – Correlation Study

---

-----> correlation coefficients, MAX for nps = 1, ..., 10000

RNG	lag= 0	1	2	3	4	5	6	7	8
0	1.0000	0.0104	0.0112	0.0105	0.0095	0.0095	0.0097	0.0104	0.0095
1	1.0000	0.0111	0.0097	0.0095	0.0112	0.0098	0.0098	0.0092	0.0099
2	1.0000	0.0093	0.0098	0.0102	0.0098	0.0101	0.0111	0.0083	0.0097
3	1.0000	0.0103	0.0093	0.0094	0.0093	0.0104	0.0099	0.0107	0.0088
4	1.0000	0.0099	0.0105	0.0098	0.0087	0.0099	0.0094	0.0103	0.0103
5	1.0000	0.0087	0.0101	0.0091	0.0096	0.0096	0.0095	0.0109	0.0086
6	1.0000	0.0100	0.0096	0.0096	0.0106	0.0111	0.0100	0.0093	0.0095
7	1.0000	0.0100	0.0086	0.0096	0.0100	0.0106	0.0102	0.0092	0.0102
8	1.0000	0.0095	0.0104	0.0107	0.0092	0.0089	0.0093	0.0117	0.0100

-----> correlation coefficients, MIN for nps = 1, ..., 10000

RNG	lag= 0	1	2	3	4	5	6	7	8
0	1.0000	-0.0093	-0.0096	-0.0097	-0.0112	-0.0094	-0.0092	-0.0097	-0.0113
1	1.0000	-0.0102	-0.0106	-0.0103	-0.0098	-0.0099	-0.0100	-0.0108	-0.0098
2	1.0000	-0.0113	-0.0105	-0.0084	-0.0094	-0.0098	-0.0105	-0.0096	-0.0106
3	1.0000	-0.0089	-0.0088	-0.0100	-0.0106	-0.0092	-0.0109	-0.0101	-0.0096
4	1.0000	-0.0100	-0.0089	-0.0097	-0.0118	-0.0100	-0.0095	-0.0104	-0.0097
5	1.0000	-0.0101	-0.0093	-0.0112	-0.0093	-0.0097	-0.0103	-0.0093	-0.0099
6	1.0000	-0.0097	-0.0098	-0.0100	-0.0093	-0.0098	-0.0089	-0.0087	-0.0103
7	1.0000	-0.0091	-0.0113	-0.0099	-0.0099	-0.0103	-0.0099	-0.0087	-0.0088
8	1.0000	-0.0093	-0.0100	-0.0116	-0.0094	-0.0111	-0.0096	-0.0093	-0.0105

# RNG #2 – Correlation Study

---

```

-----> cross-correlation, MAX for nps = 1, ..., 10000
RNG lag= 0      1      2      3      4      5      6      7      8
0-0  1.0000  0.0104  0.0112  0.0105  0.0095  0.0095  0.0097  0.0104  0.0095
0-1  0.0104  0.0091  0.0112  0.0102  0.0096  0.0100  0.0103  0.0089  0.0102
0-2  0.0123  0.0109  0.0109  0.0094  0.0102  0.0103  0.0091  0.0101  0.0102
0-3  0.0101  0.0094  0.0108  0.0089  0.0103  0.0088  0.0093  0.0107  0.0097
0-4  0.0099  0.0116  0.0107  0.0087  0.0107  0.0088  0.0095  0.0099  0.0096
0-5  0.0101  0.0093  0.0092  0.0102  0.0100  0.0096  0.0098  0.0096  0.0088
0-6  0.0091  0.0097  0.0105  0.0099  0.0108  0.0113  0.0095  0.0098  0.0104
0-7  0.0094  0.0095  0.0099  0.0098  0.0090  0.0094  0.0094  0.0091  0.0112
0-8  0.0094  0.0098  0.0107  0.0090  0.0097  0.0092  0.0088  0.0106  0.0122

```

```

-----> cross-correlation, MIN for nps = 1, ..., 10000
RNG lag= 0      1      2      3      4      5      6      7      8
0-0  1.0000 -0.0093 -0.0096 -0.0097 -0.0112 -0.0094 -0.0092 -0.0097 -0.0113
0-1 -0.0089 -0.0089 -0.0103 -0.0099 -0.0089 -0.0091 -0.0100 -0.0113 -0.0095
0-2 -0.0090 -0.0111 -0.0091 -0.0106 -0.0090 -0.0106 -0.0096 -0.0097 -0.0109
0-3 -0.0098 -0.0104 -0.0094 -0.0095 -0.0090 -0.0118 -0.0095 -0.0101 -0.0113
0-4 -0.0099 -0.0098 -0.0095 -0.0094 -0.0094 -0.0102 -0.0106 -0.0089 -0.0119
0-5 -0.0101 -0.0100 -0.0092 -0.0083 -0.0111 -0.0094 -0.0107 -0.0100 -0.0098
0-6 -0.0092 -0.0095 -0.0094 -0.0109 -0.0089 -0.0097 -0.0092 -0.0091 -0.0087
0-7 -0.0090 -0.0095 -0.0103 -0.0103 -0.0093 -0.0100 -0.0096 -0.0096 -0.0094
0-8 -0.0105 -0.0095 -0.0091 -0.0090 -0.0124 -0.0099 -0.0127 -0.0100 -0.0088

```

# Summary

---

- **The current MCNP RNG can handle up to  $6 \times 10^{13}$  histories without repetition using RNG #2, #3, or #4**
- **RNG #2 has excellent statistical properties, in fact better than the traditional 48-bit default generator**
- **Using different adders, RNG #2 could be used to provide different, independent RN sequences for different classes of events in the MC simulation, possibly resolving reproducibility problems**
- **Thorough examination & testing of RNG #2 demonstrates that using different adders does not introduce correlation.**
  - **Different adders could be used for different particle types**
  - **Different adders could be used for sources, tracking, tallies, etc.**



**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Monte Carlo Codes – Basic Structure & Algorithm**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Outline

---

- **Perspective**
- **Building a Monte Carlo Code**
  - **Basic building blocks**
  - **Testing**
  - **Data Structures**
  - **Overall code organization**
  - **Random walk for a history**

# Perspective

---

- **The lectures on computing & Monte Carlo codes are intended for both future code developers and code users**
- **For future code developers**
  - Follow these guidelines until you're good enough to make your own
  - Throw out your “Numerical Recipes” book - look at real codes & the literature
  - Learn both Fortran & C, and perl & python & bash scripting
- **For code users**
  - A general idea of what the codes do & why helps in deciphering input manuals & output results
  - Need to be good at using editor & shell windows & command line, not just point-and-click GUIs
  - Never just accept the MC results - always question whether results are reasonable & what you expect

# Building a Monte Carlo Code

# Basic building blocks

---

- **Random number generator**
  - Use a known, well-tested RNG - MCNP routines
- **Random sampling routines**
  - See my notes, Devroye's book, Kahn's report, 3rd MC Sampler, ...
- **Geometry routines - locate, distance, neighbors, boundary**
  - For mesh geometry & very simple 3D, can do it yourself
  - For general 3D, this is a career - borrow from real codes
- **Physics routines - access, search, interpolate, sample**
  - For 1-group or multigroup, do it yourself
  - For general continuous-energy, borrow from real codes
- **Tally & statistics routines**
  - Usually straightforward, but review your statistics

# Testing a Monte Carlo Code

---

- **Basic building blocks must be tested separately, before putting into larger code**
  - RN generator
  - Random sampling routines
  - Distance calculations
  - Table search routines
  - Interpolation routines
- **Whole code must be tested on as many problems as possible where correct answers are known**
  - Analytical problems, with exact solutions
  - Experiments, with measured results
    - Be wary of experiment error bars & model uncertainties
    - Calculate many experiments, never just one

# General Guidelines

---

- **For any scientific & engineering programs, always use “double-precision” for real numbers**

Fortran:                    **real(8) x**  
 C/C++:                    **double x;**  
 Matlab:                    **(default is double)**

- **Data types should be explicit for constants**

Fortran:                    **pi = 3.14159265358979d+0**  
                               **not pi = 3.14159265358979**

- **Integer lengths - 32-bit vs 64-bit**

Fortran:    **integer id           integer(8):: id   integer(8):: id**  
 C/C++:     **int id;               long id;           long long id;**

Usually, Fortran integer & C int limited to:  $\leq 2,147,483,647$

OK for simple demo codes; production codes usually need bigger ints

Matlab:       **uses real(8)**

# Data Structures (1)

---

- **Particle - minimum attribute set**

```
struct  particle {  
    long long      id;           // particle identifier number  
    double         x,y,z;       // position  
    double         u,v,w;       // direction cosines,  $u^2 + v^2 + w^2 = 1$   
    double         e;           // energy (or group number, integer)  
    double         wgt;        // weight  
    long long      seed;        // RN seed - most codes don't do this!  
}
```

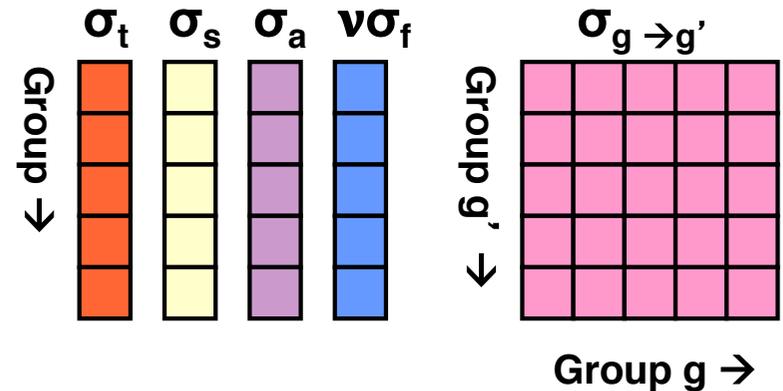
- **For convenience & speed, often include derived info:**

```
long      cell;           // current cell number  
double    dcol;          // distance to collision  
double    dsur;          // distance to cell boundary surface  
long      jsur;          // number/label of boundary surface  
long      ix,jx,kx;      // lattice cell index numbers  
.....
```

## Data Structures (2)

- **Multigroup Cross-sections**

- Vectors of  $\sigma'$  s
- Matrix of group-to-group scatter
- Group 1 - highest energy range



- **Continuous-energy Cross-sections**

- Complex format, 68 page description in MCNP Manual Vol-III
- Microscopic  $\sigma'$  s given as ladder of  $(E_k, \sigma_k)$  pairs or sets
  - $\sigma_k$  is the cross-section at energy  $E_k$
  - For  $E_k < E < E_{k+1}$ , linear interpolation [sometimes lin-log, log-lin, log-log]

$$f = \frac{E - E_k}{E_{k+1} - E_k}$$

$$\sigma_t(E) = (1 - f) \cdot \sigma_{t,k} + f \cdot \sigma_{t,k+1}$$

- Data for scattering laws has varied, complex formats

# Overall Code Organization

---

## Initialize problem

- read input, or hard-wired setup - geom, xsecs, options
- clear tally arrays for problem
- set RN seed for problem

## Do n=1, nhistories

### Initialize history

- clear tally arrays for history
- set RN seed for history

### Source for history n

- set x,y,z, u,v,w, E, wgt, cell

### Random walk for history n

- geometry, physics, tallies for history

### Statistics

- add history tallies & tallies<sup>2</sup> to problem tallies

## end-of-history-loop

## Compute overall results & statistics

# Random Walk for a History

---

Source - set  $x,y,z$ ,  $u,v,w$ ,  $E$ ,  $wgt$ ,  $cell$

Do while  $wgt > 0$

- get material number & xsec data
- $dist\_collision$  from random sampling
- $dist\_boundary$  from distance routines

$dist = \min( dist\_collision, dist\_boundary )$

$(x,y,z) = (x,y,z) + dist * (u,v,w)$

- make pathlength tallies

if(  $dist\_collision < dist\_boundary$  )

- collision physics, get new  $u,v,w$ ,  $E$ ,  $wgt$
- make collision tallies
- if particle terminated, exit loop

else

- boundary routine
- find neighbor cell
- make surface tallies
- if particle escapes, exit loop

- Russian roulette & splitting games

end-of-flight/collision-loop



**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Code Development – How to Time & Test**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Introduction

---

- **For a production-level code**
  - **Correct results is #1**
    - Compare code results to experiments or analytic solutions
    - Document the verification/validation results
  - **Run-time is #2**
    - If calculations take too long, users might not do them or might take undesirable shortcuts...
    - On today's computers, parallelism is required for decent performance
- **For developing new algorithms, methods, & numerical schemes**
  - **Generally done stand-alone, separate from large production code**
  - **Developers need to test & time the old vs new approaches**
  - **Impact on production code runtime depends on how often the new coding is used, and also on the applications**

# Timing

# How to Time

---

- **Small sections of coding may take  $\mu$ sec or nsec**
  - **For such short times, system timing routines are not reliable for a single execution of the coding**
  - **Need to take an average execution time for many runs**
    - May need 1000s or Ms of repetitions for reliable timing
    - Subtract the overhead for repeating the test
    - Need to vary the inputs for the tests, perhaps randomly
  - **For threaded coding**
    - Timing should be in terms of wall-clock time, not cpu-time
  - **For single-thread coding**
    - Either cpu-time or wall-time is fine
    - cpu-time is easier
    - In MC, threading is by history, so any coding acting on only 1 history (particle) should be timed for a single thread

# Fortran 2003 CPU Timing

---

```
call cpu_time ( t )
```

To measure cpu time:

```
real(8) :: t, t1, t2
call cpu_time( t1 )
do k=1,nrepeat
    .....code being timed
enddo
call cpu_time( t2 )

!==> time/trial
t = (t2 - t1)/nrepeat - t_overhead
```

# Fortran 2003 Wall-clock Timing

---

call **system\_clock**( COUNT= count, COUNT\_RATE= crate, COUNT\_MAX= cmax )

count, crate, cmax: integers with the same KIND attribute

To measure elapsed wall-clock time:

```
integer(8) :: count1, count2, crate, cmax
call system_clock( COUNT=count1 )
    .....code being timed
call system_clock( COUNT=count2, &
    & COUNT_RATE=crate, COUNT_MAX=cmax )
t = (count2-count1) / real(crate,8)
! in case count rolls over:
if( t<0 ) t = t + cmax/real(crate,8)
```

Note for Intel Fortran-17, MacOS 10.12 :

- Using integer(4): cmax/crate ~ 2.5 days, max interval
- Using integer(8): cmax/crate ~ 300K years, max interval

# Timing Example

---

```
nrepeat = 1000000                                <--- should be large,
                                                so total time is > a few seconds

!===> get overhead per trial
call cpu_time(t1)
s = 0
do j=1,nrepeat
  s = s + rang()                                <--- include overhead, & some extra
                                                (cheap) op so that compiler has
enddo                                           to do something & can't optimize
call cpu_time(t2)                               everything away
t_overhead = (t2-t1) / nrepeat

!===> timing for binary search
call cpu_time(t1)
do j=1,nrepeat
  r = rang()
  k = bsearch( npts, cdf, r )
enddo
call cpu_time(t2)
t = (t2-t1)/nrepeat - t_overhead

write(*,*) "bsearch:", t, "sec/trial"
```

# Timing & Scaling

---

- **For many algorithms, the time/trial depends on the size of a dataset**
  - Table searches
  - Permutations
  - Reordering data
  - Sampling from a discrete PDF
- **Timing tests need to be performed with different dataset sizes**
  - The "best" algorithm for small datasets may be bad for large datasets
  - Plots of (time/trial) vs (dataset size) are especially useful to identify which algorithms are best for a range of likely dataset sizes

# Timing Example – with Scaling

```
nrepeat = 1000000
```

```
!====> get timing overhead/trial
```

```
call cpu_time(t1)
```

```
s = 0
```

```
do j=1,nrepeat
```

```
  s = s + rang()
```

```
enddo
```

```
call cpu_time(t2)
```

```
t_overhead = (t2-t1) / nrepeat
```

```
!====> timing for various table sizes
```

```
n_ndata = 10
```

```
ndata = [ 2, 4, 8, 16, 32, &  
& 64, 128, 256, 512, 1024 ]
```

```
do k=1,n_ndata
```

```
  n = ndata(k)
```

```
  write(*,*) "Test tablesize =",n
```

```
  call make_data( n, x, pdf, cdf )
```

```
  call time_bsearch( nrepeat, n, &  
    & cdf, t_overhead)
```

```
enddo
```

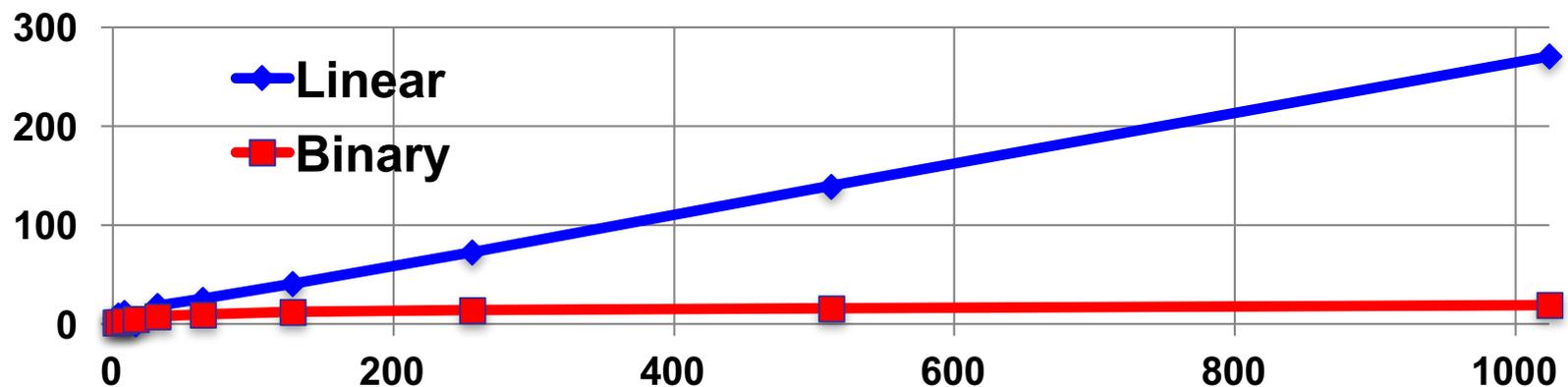
```
subroutine time_bsearch( nrepeat,npts,cdf,tover)  
  integer,intent(in) :: nrepeat, npts  
  real(8),intent(in) :: cdf(:), tover  
  integer :: j,k  
  real(8) :: r, t, t1, t2  
  !====> timing for binary search  
  call cpu_time(t1)  
  do j=1,nrepeat  
    r = rang()  
    k = bsearch( npts, cdf, r )  
  enddo  
  call cpu_time(t2)  
  t = (t2-t1)/nrepeat - tover  
  write(*,*) "bsearch:", t, "sec/trial"  
end subroutine time_bsearch
```

```
subroutine make_data( n, x, pdf, cdf )  
  !====> randomly create piecewise linear PDF  
  integer,intent(in) :: n  
  real(8),intent(out) :: x(:), pdf(:), cdf(:)  
  integer :: k  
  do k=1,n  
    x(k) = k  
    pdf(k) = rang()  
  enddo  
  cdf(1) = 0.0  
  do k=2,n  
    x1=x(k-1); x2=x(k);  
    p1=pdf(k-1); p2=pdf(k)  
    cdf(k) = cdf(k-1) + 0.5*(p2+p1)*(x2-x1)  
  enddo  
  pdf(1:n) = pdf(1:n) / cdf(n)  
  cdf(1:n) = cdf(1:n) / cdf(n)  
end subroutine make_data
```

# Timing Example – 6 Variations on Table Searches

- For randomly generated PDFs & randomly sampled targets
- Lookup time (nanosec) vs number of bins:

N bins =	2	4	8	16	32	64	128	256	512	1024
Linear	6.0	9.2	11.7	13.4	18.2	25.2	40.4	72.4	139.4	270.7
Binary	1.6	2.6	4.3	6.7	9.3	11.6	14.6	18.0	21.5	24.9
Binary, shift	1.4	2.2	3.5	5.4	7.3	9.2	12.2	14.1	15.7	19.0
Binary, shift, merge	1.4	2.2	3.5	5.3	7.6	9.4	12.4	14.4	16.2	19.1
Binary, goto	1.3	2.2	3.4	5.6	7.3	9.2	12.2	14.1	15.7	19.1
Binary, no if-tests	2.6	3.6	5.0	6.6	8.2	9.8	11.7	13.8	15.7	17.9



MacBook Pro  
3.5 GHz I7  
2.1 Ghz LPDDR3

# Testing

# Introduction

---

- **General ways to test new algorithms**

- **Compare 2 or 3 different approaches**

- For searching, may want to compare results using linear & binary searches
- For sampling, may want to compare rejection & direct, old vs new, . . . . .

- **For random sampling algorithms, common approaches are**

- Use very many **histogram bins** (1000s) for sampled results, compare to original PDF on same bin structure
- Compute **moments** of sampled results & compare to analytic moments

$$m_1 = \text{sum}(x_i)/N, \quad m_2 = \text{sum}(x_i^2)/n, \text{ etc.}$$

- **Need to repeat the sampling or searching algorithm very, very many times, varying the tables or probabilities, both size & shape**

- Sometimes, need to pay particular attention to end-cases

# Example – Searching

---

- Compare different methods & check

```
nrepeat = 10000000

!===> pick different table sizes
n_nstable = 10
ntable = [ 2, 4, 8, 16, 32, 63, 64, 65, 128, 256 ]

do k=1,n_nstable
  n = ntable(k)

  do k=1,nrepeat
    !---> generate a random PDF
    call make_random_pdf( n, pdf )

    !---> search different ways (with same RN)
    r = rang()
    i = linear_search( n,cdf, r )
    j = binary_search( n,cdf, r )

    !---> check
    if( i /= j ) stop '***** error 1 *****'
    if( r < pdf(i) ) stop '***** error 2 *****'
    if( r > pdf(i+1) ) stop '***** error 2 *****'
  enddo
enddo
```

# Example – Sampling, Histogram Bins

---

- Histogram binning

```
nbins = 1000
```

```
xmax = ... Max value for range
```

```
xmin = ... Min value for range
```

```
!---> fill reference array with exact pdf at bin midpoints
```

```
call fill_with_exact_pdf( nbins, x_exact )
```

```
nrepeat = 1000000
```

```
!---> repeated sampling (may have outer loop for parameters)
```

```
do k=1,nrepeat
```

```
    x = sample_pdf()
```

```
    bin = 1 + nbins*(x-xmin)/(xmax-xmin)
```

```
    x_sample(bin) = x_sample(bin) + 1
```

```
enddo
```

```
x_sample = x_sample / nrepeat
```

```
!---> compare x_exact(:) to x_sample(:) ...
```

# Example – Sampling, Moments

---

- Compute moments of exact PDF & sampled PDF

```
nmom = 10
```

```
!---> fill reference array with mementos of exact pdf  
call fill_with_exact_moments( nmom, moments_exact )
```

```
nrepeat = 1000000
```

```
!---> repeated sampling (may have outer loop for parameters)
```

```
do k=1,nrepeat
```

```
  x = sample_pdf()
```

```
  !!--> compute sample moments
```

```
  do j=1,nmom
```

```
    moments_sample(j) = moments_sample(j) + x**j
```

```
  enddo
```

```
enddo
```

```
moments_sample = moments_sample / nrepeat
```

```
!---> compare moments_exact(:) to moments_sample(:) ...
```

# Assessing Test Results

---

- **Plots of exact vs sampled results**
- **Compute goodness-of-fit parameters**
  - RMS difference, max difference, etc.
  - Could compute statistics on sampled results
- **For moment checking**
  - Could compute statistics on sampled moments
  - RMS differences, etc.
- **For modern unit-testing, need to decide on definite pass/fail criteria**

# Final Comments

---

- **Timing & testing takes a lot of time & effort**
  - Sometimes more time than the algorithm development
  - It's work, not fun
  - Necessary – if not done, the new method or algorithm is worthless
- **Documenting the work**
  - **Modern code development practices**
    - Software Quality Assurance (SQA)
    - Rigorous SQA is required by many professional standards
    - Must document
      - Basis for method (ie, theory, algorithm, ...)
      - Testing results
      - Timing/scaling results not required (but should be)
      - Independent review
  - **It's work, not fun**
  - **Necessary – if not done, the new method or algorithm is worthless**
- **By today's standards, if code development is not documented, tested, & reviewed, it won't be used**







**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Vector & Parallel Monte Carlo**

**From: F.B. Brown & T.M. Sutton, "Monte Carlo Fundamentals", KAPL-4823, 1996**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Introduction

---

## Particle transport Monte Carlo is naturally parallel

- Fixed-source problems: each particle in **problem** is independent
  - Eigenvalue problems: each particle in **generation** is independent
- ⇒ Particle **histories** can be analyzed in parallel

## Monte Carlo is often the first use for advanced computers

- Easy to port — compact coding, little I/O, simple parallel algorithm
- Flexible — independent histories on each node
- Big payoff — bigger & faster calculations

## Computational considerations

- Expensive — hours / days / weeks of computing
- Compact — moderate memory size
- CPU-intensive — very little I/O or communications

***T( computation ) >> T( communications )***

# Vector Processing

## Vector Processing

- Fortran

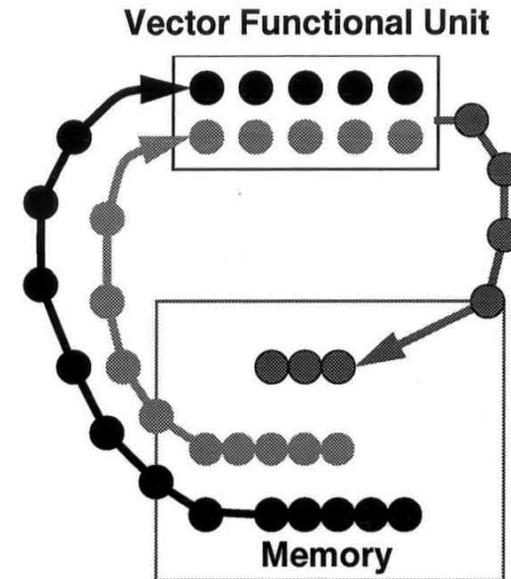
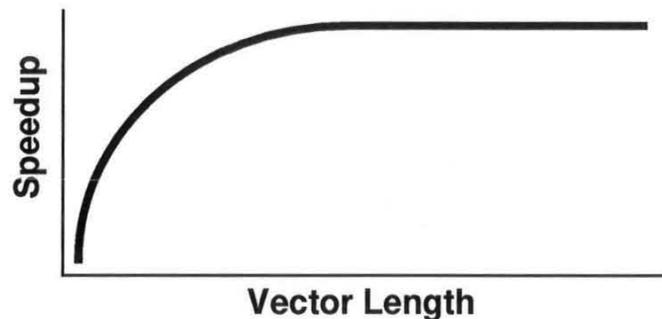
```
do j=1,L
  a(j) = b(j) + c(j)
enddo
```

- Timing

$$T_{\text{vector}} = t_{\text{startup}} + L \cdot t_{\text{operation}}$$

- Speedup

$$S = T_{\text{scalar}} / T_{\text{vector}} = \frac{L \cdot t_{\text{scalar-op}}}{t_{\text{startup}} + L \cdot t_{\text{vector-op}}} \Rightarrow \frac{t_{\text{scalar-op}}}{t_{\text{vector-op}}}$$



# Vector Processing

---

## Vector Operations

- **Gather** — form a contiguous vector from data in arbitrary locations

```

do j=1,L
    b(): 7, 5, 3, 6, 1, 14
    i(): 4, 1, 2, 4, 4
    a(j) = b( i(j) )
enddo
a(): 6, 7, 5, 6, 6

```

- **Scatter** — disperse vector data to arbitrary locations

```

do j=1,L
    b(): 1, 2, 3, 4
    i(): 4, 1, 2, 5
    a( i(j) ) = b(j)
enddo
a(): 2, 3, ?, 1, 4, ?, ...

```

# Vector Processing

---

## Vector Operations

- **Mask** — either/or selection of data from two vectors

do j=1,L	
if( test(j) ) then	test(): T, T, F, F, T
c(j) = a(j)	a(): 0, 1, 2, 3, 4
else	b(): 5, 6, 7, 8, 9
c(j) = b(j)	
endif	
enddo	c(): 0, 1, 7, 8, 4

- **Compressed Index Generation** — find the indices of selected items in a vector

k = 0	
do j=1,L	test(): T, T, F, F, T
if( test(j) ) then	
k = k + 1	
indx(k) = j	
endif	k: 3
enddo	indx(): 1, 2, 5

# Vector Processing

---

## The Tally Loop (scalar)

- Indexed accumulation

```
do j=1,L
    sum( i(j) ) = sum( i(j) ) + r(j)
enddo
```

- Used to tally particle scores into bins, for overall results
- Tally operations account for 1-10% of time
- **Not** readily vectorized (some tricks for Cray-C90)

# Vector Processing

---

## Writing Efficient Vector Coding

- Clean loops — structure & indent (Good-looking code runs faster!)
- Innermost loop should be the vector loop
- Avoid IF tests, unless strictly "either/or"
- Never use "GO TO" statements
- No subroutine calls
- No user-defined function calls
- No recursion (ie, forward-stores or backward-fetches)
- Timing estimates:
  - Count **all** operations inside loop, including **both** branches for IF's.
  - Multiply by vector length & clock cycle time.
  - Measure. If much different from estimate, find out why !

# Vector Processing

---

## Amdahl's Law

- If a computation has fast (vector) & slow (scalar) components, the overall calculation time will be dominated by the slower component

- Speedup =  $\frac{1}{(1 - f) + f/R}$ ,

where  $f$  = fraction vectorized  
 $R$  = max speedup from vector

for $R = 10$				for $R = \infty$			
<u>f</u>	<u>S</u>	<u>f</u>	<u>S</u>	<u>f</u>	<u>S</u>	<u>f</u>	<u>S</u>
20%	1.2	90%	5.3	20%	1.3	90%	10
40%	1.6	95%	6.9	40%	1.7	95%	20
60%	2.2	99%	9.2	60%	2.5	99%	100
80%	3.6	99.5%	9.6	80%	5	99.5%	200

- For effective vector performance, **must vectorize everything !**

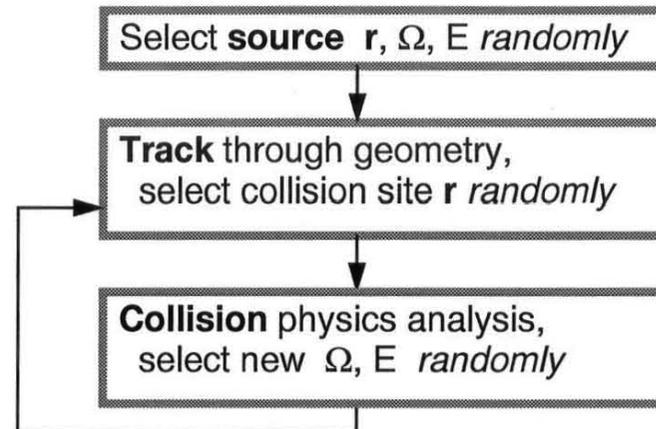
# Vectorization

---

## Vectorized Monte Carlo

- **Monte Carlo**

Simulate neutron behavior  
by random-walk.



- **Conventional Monte Carlo**

Analyze many events for one neutron, repeat for other neutrons

- **Vectorized Monte Carlo**

Analyze many neutrons for one event, repeat for other events

⇒ **Event-based algorithms** developed by Kalos, Brown/Martin, Bobrowicz

# Vectorization

---

## Monte Carlo is difficult to "vectorize"

- Branching, data retrieval, & arithmetic operations vary for each particle, depending on location, type of collision, code options, etc.
- Typically,  $\sim 1/3$  of essential Fortran statements are IF-tests, which inhibit vectorization
- Not useful:
  - "automatic vectorizers"
  - syntactic hand-vectorization by programmers

[ In early 1980s, LANL tried each approach with *mcnp*  $\rightarrow$  2X *slower* ]

## Method for Vectorizing Monte Carlo

1. Use supercomputer with vector hardware for data-handling
2. Deliberate & careful development
3. Restructure the database
4. Restructure & rewrite the Monte Carlo code

# Vectorization

---

## Method for Vectorizing Monte Carlo

### 1. Use supercomputer with vector hardware for data-handling

- Only ~40% of operations are floating-point arithmetic (\*, +, -, /, sqrt)
- 40-60% of operations involve data-handling, indexing, selection, .....
- Must have hardware support for data-handling (gather, scatter, mask, compressed index, ...)

### 2. Deliberate & careful development

- Start small, with few options
- No committees !!!
- Focus effort on total vectorization
- Build gradually, restructuring as needed for new features
- Debugging is extremely difficult — test everything, separately & integrated

### 3. Restructure the database

- Unified data formats, with no special cases
- Arrange for simple & logical direct addressing using vector gather operations
- Use some new (but equivalent) physics, if necessary

### 4. Restructure & rewrite the Monte Carlo code

- "Top-down" development, based on **event-driven algorithm**
- Use some new (but equivalent) physics, if necessary
- Avoid rejection methods for random sampling
- "Vectorize" the IF-tests by data motion, extra computation, or new algorithms

# Vectorization

---

## Vectorizing IF-tests

In Monte Carlo codes, IF-tests arise in the context of:

implicit loops, conditional coding, code options

## Implicit loops

- Logic of the form "loop UNTIL ....."
- Usually coded as "IF ..... GOTO" & backward branch, instead of "DO ....."
- Number of passes is generally not known in advance
- Some particles satisfy the exit conditions on first pass, others take many passes
- Vectorize by:
  - Data motion — rearrange the particle data after each pass (eg, event-driven algorithm)
  - Extra computation — dummy ops on "finished" particles till all are done
  - Different math/physics — eliminate implicit loop (eg, direct sampling instead of rejection)

## Conditional coding

- Selective operations of some particles, but not others
- Vectorize by:
  - Gather / Operate / Scatter
  - Rearrange selective ops into series of "either/or" ops using vector masks
  - Generalized equations, without special cases

## Code options

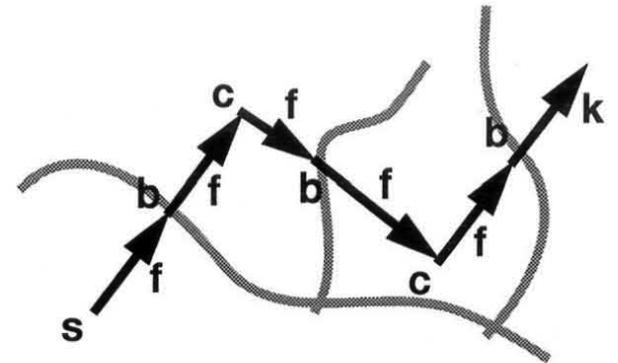
- Easy — one test/branch for all particles

# Event-Driven Algorithm

## Monte Carlo — Vectorization

### Conventional Algorithm

<i>history</i>	<i>event sequence</i>
1	s f c f b f b f c f b k
2	s f b f c f c f b f b f b k
3	s f b f b f b k
4	s f b k
5	s f c f c f c f b f b f c f b f b k



### Event-Based Algorithm

<i>history</i>	<i>event sequence</i>
1	s f ~ c f b ~ f b ~ f ~ ~ ~ ~ c f b k
2	s f b ~ f ~ c f ~ c f b f b f ~ ~ b k
3	s f b ~ f b ~ f b ~ ~ ~ ~ ~ ~ ~ ~ k
4	s f b ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ k
5	s f ~ c f ~ c f ~ c f b f b f c f b ~ f b k

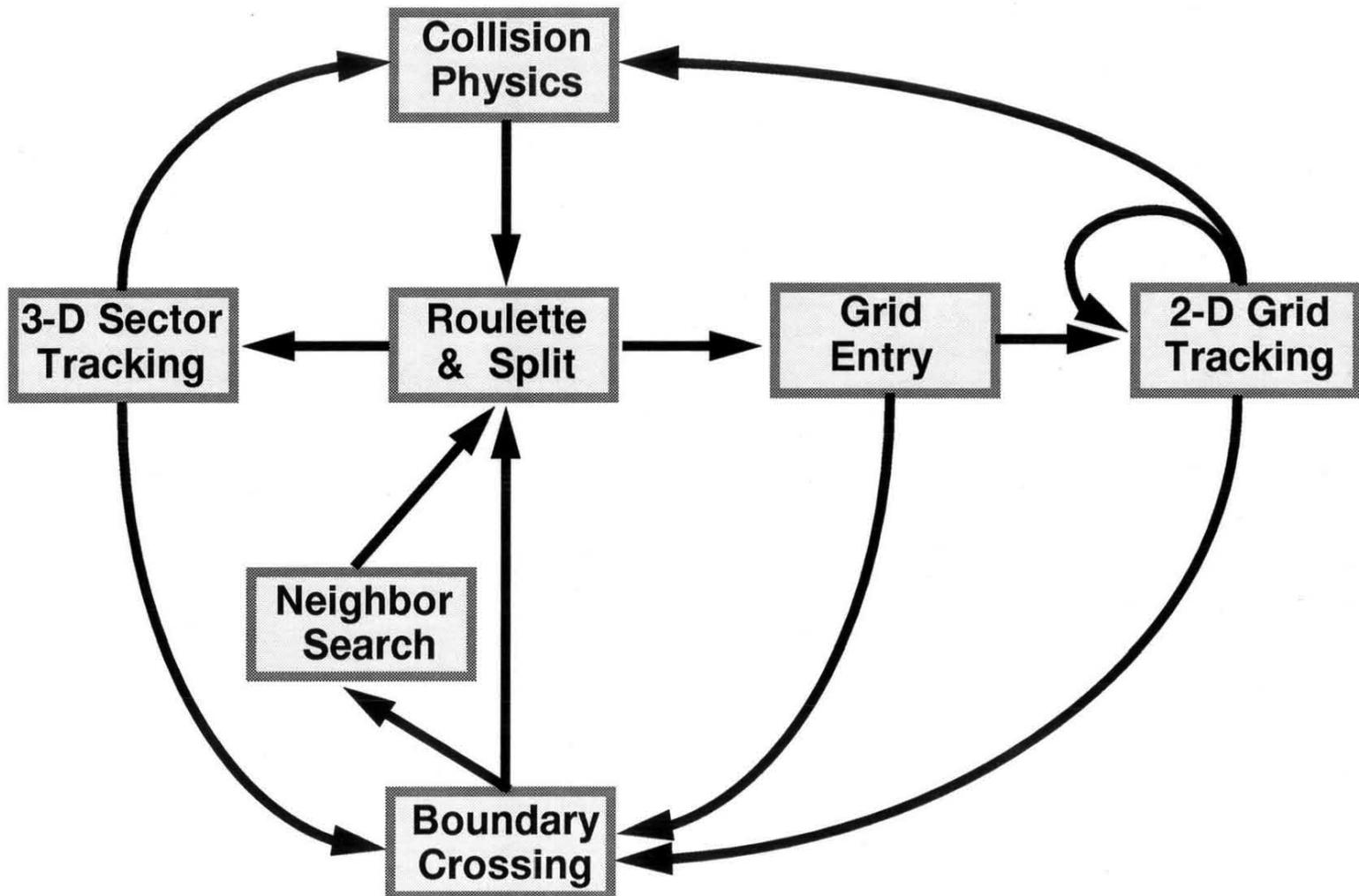
*vector event sequence*

s f b c f b c f b c f b f b f c f b k f b k
---

# Event-Driven Algorithm

---

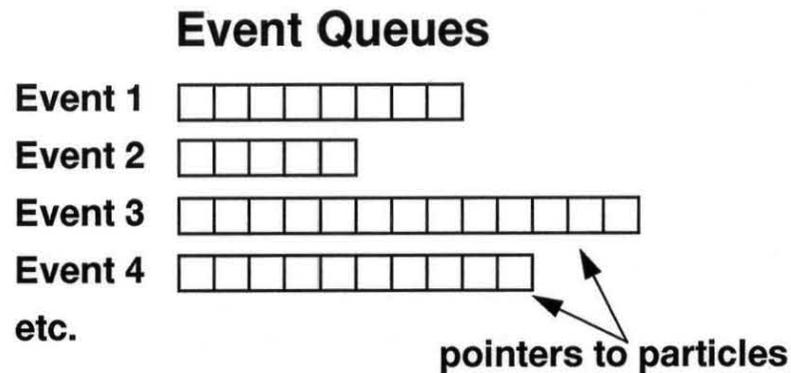
## RACER Computational Events



# Event-Driven Algorithm

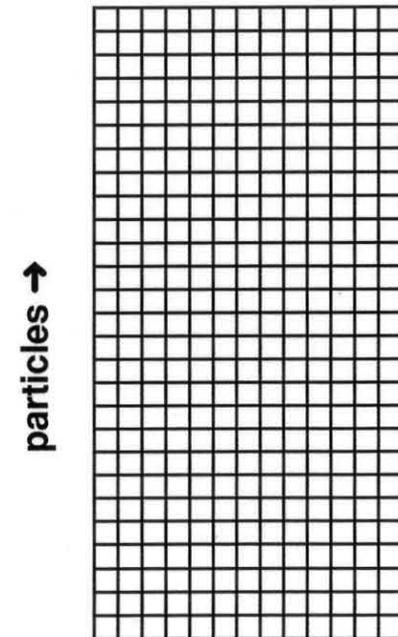
## Monte Carlo — Vectorization

### Event-Driven Algorithm:



### Particle Stack

Attributes →  
 $x, y, z, u, v, w, E, \dots$

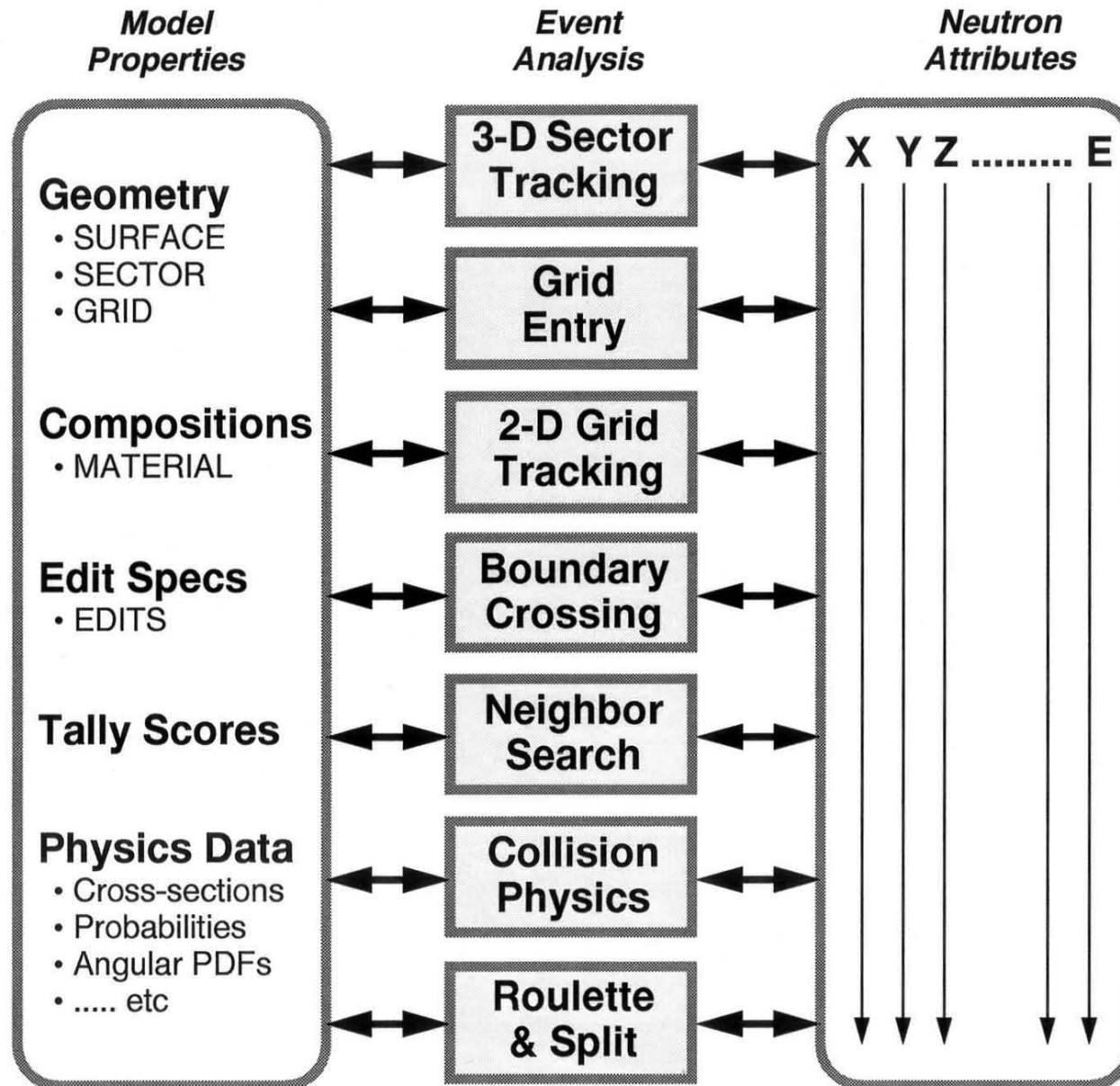


while events are pending:

- — **select event** (1,2,3,...) with largest event-queue
- — **execute the event:**
  - **Pull**  $N$  **pointers** from event-queue
  - **Gather attributes** for  $N$  particles from stack
  - **Analyze** event — vector calculation
  - **Scatter** modified **attributes** to stack
  - **Push pointers** onto next event-queue
- • •

# Event-Driven Algorithm

## RACER — Code Organization



# Event-Driven Algorithm

- \* Loop over *timesteps*
  - \* Loop over *batches*
    - • → **select** starters
    - • \* Loop over *edit-groups*
      - • • → **clear** edit-group tallies
      - • • \* Loop over *super-groups*
        - • • • → get  $\sigma$ 's,  $f(\mu)$ 's, .....
        - • • • → **clear** event-queues
        - • • • → **push** to event-queues: pointers to neutrons in supergroup
        - • • • \* Loop until **event-queues** are empty
          - • • • • → **select event** with longest pending queue
          - • • • • → **pull** from event-queue: pointers to neutrons
          - • • • • → **gather**: needed neutron attributes
          - • • • • → **analyze event & tally**
          - • • • • → **scatter**: updated neutron attributes
          - • • • • → **push** to next-event queue: pointers to neutrons
          - • • • • ...
          - • • • • ...
          - • • • • ...
      - • • • • → update eigenvalue, results, stats
      - • • • • ...
  - • • • • → depletion
  - • • • • ...

# Vectorization

---

## Monte Carlo — Vectorization

### Status

- Vectorized Monte Carlo, with event-driven algorithms, was proven to work effectively in ~**1980** on Cray-1 & Cyber-205
- **Large speedups** (20x or more) were demonstrated in production Monte Carlo codes, on real problems
- Relatively easy to migrate to MIMD or mixed MIMD/SIMD architectures
- **Very few** large production codes have adopted this approach

### What's the problem ?

- Must restructure the entire database & rewrite the entire code
- Large amount of people-time → expensive
- "Why change something that works?" → QA work

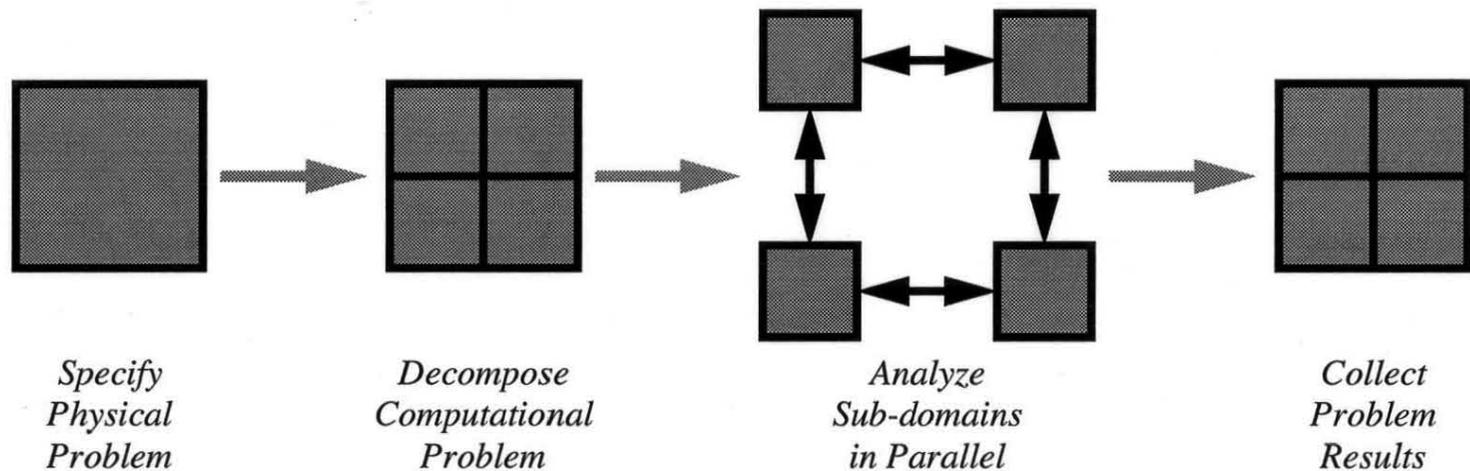
# Parallel Processing

## Parallel Programming

### SIMD Machines

- Fine-grained parallelism, low-level
- Vector algorithms & programming

### MIMD Machines & Distributed Systems



- Coarse-grained parallelism, high-level
- Ideal for loosely-coupled machines & message-passing libraries  
*pvm, p4, MPI, express, lam, parmacs, .....*

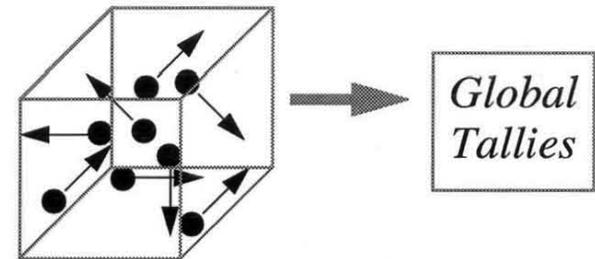
# Parallel Processing

## Physical Problem

- 3-D geometry, continuous-energy physics
- *vim* — ANL, *racer* — KAPL, *mcnp* — LANL

## Conventional Solution Algorithm

- Random walk for neutrons
- Tally events of interest

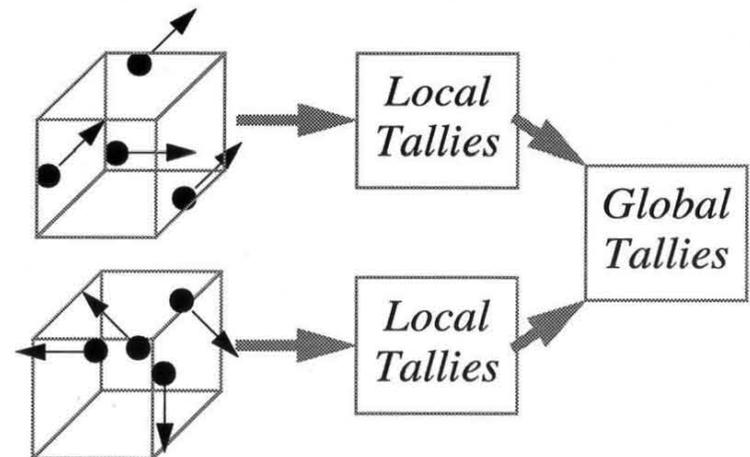


## Parallel Algorithm

- Distribute neutrons to different processors
- Local tallies on each processor
- Combine local tallies into global results

## Status

- *racer* — in production use, Cray-C90, Cray-YMP, Meiko-CS1
- *vim* — in testing, workstation network & IBM-SP1
- *mcnp* — in production use, workstation network, Cray-YMP

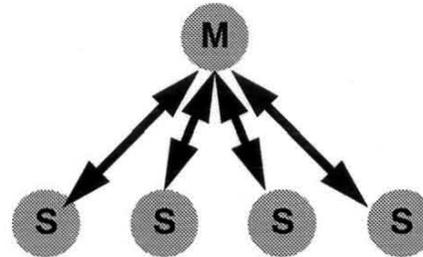


# Parallel Processing

## Monte Carlo — Parallel

### Master-slave approach

- Master
  - Control
  - All I/O
- Slaves
  - computations



### Master process

- \* Loop over *timesteps*
  - \* Loop over *batches*
    - • → select starters
    - • \* Loop over *chunks*
      - • • → *find a ready slave*
      - • • → *send starters to slave*
      - • • ...
    - • \* Loop over *slaves*
      - • • → *collect slave tallies*
      - • • ...
    - • • → update eigenvalue, results, stats
    - • ...

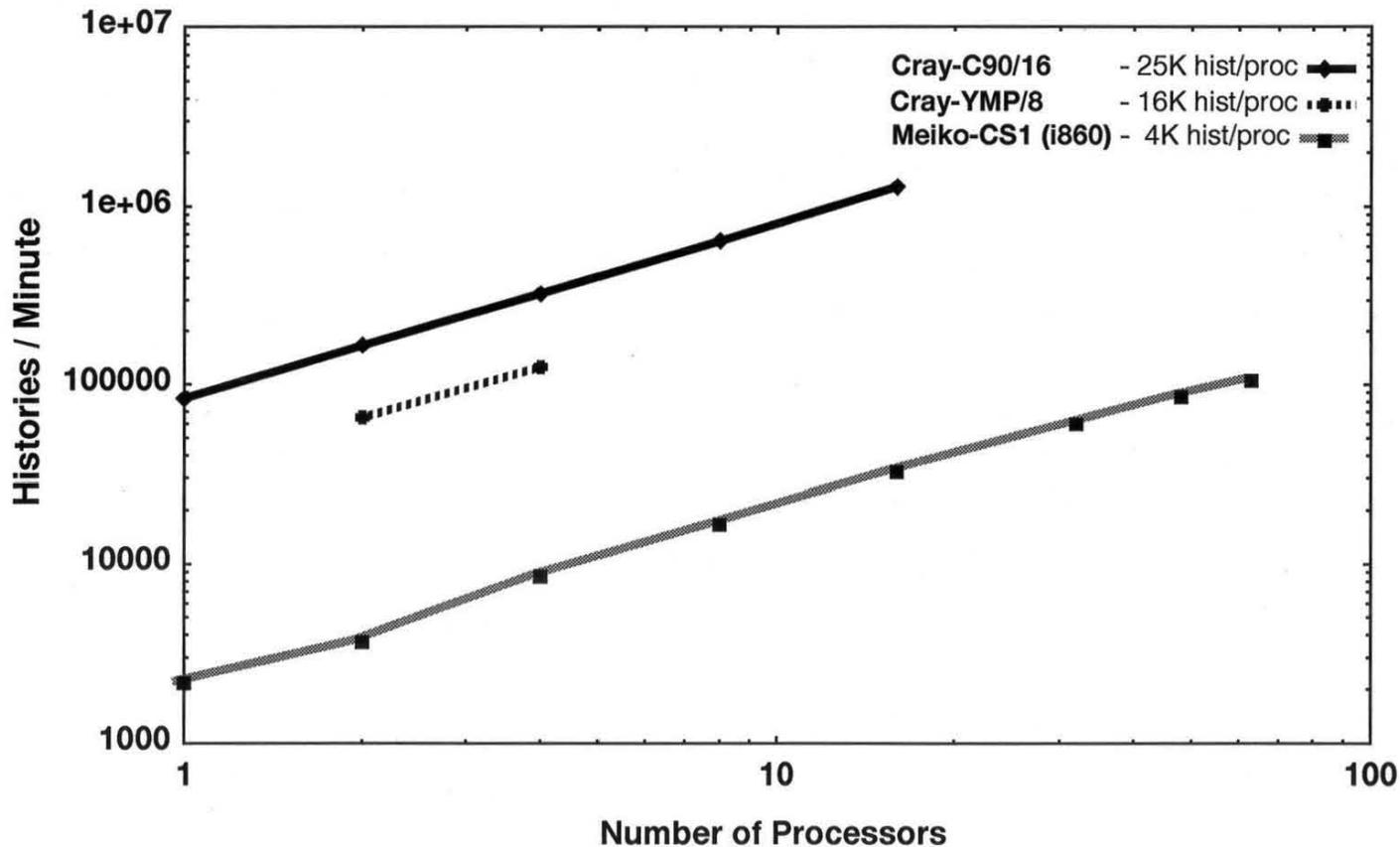
### Slave process

- \* Loop *forever*
  - → initialize tallies
  - \* Loop over *chunks*, till done
    - • • → *tell master: ready for work*
    - • • → *receive starters from master*
    - • • \* Loop over *histories* in chunk
      - • • • → *random walk* for 1 history
      - • • • ...
    - • • ...
  - • • → *send tallies to master*
  - • ...

# Performance

## *racer* — Parallel / Vector Performance

- Measured performance → histories / minute
- 3-D full-core PWR test problem
- Fixed number of histories/processor
- **Range in performance spans 1000x** [8 hr Cray-C90] ~ [1 yr workstation]



# Issues

---

## Vector & Parallel Monte Carlo — Issues

- **Hierarchical parallelism**
- **Shared memory vs distributed memory**
- **Parallel speedup & scaling**
- **Software & Portability**
- **Reproducibility**

# Monte Carlo Algorithms: Vector & Parallel

## Scalar Monte Carlo

- Loop over *batches*
  - 
  - Loop over *particles*
    - 
    - analyze many events for 1 particle history
    - ...
  - ...

## Parallel Scalar Monte Carlo

- Loop over *batches*
    - Loop over *particles*, N at a time
      - • scalar scalar ... scalar
      - • cpu-1 cpu-2 ... cpu-N
      - ...
      - ...
- parallel ⇒ 1 particle per CPU,  
scalar analysis
- high-level: parallel  
low-level: scalar

## Vector Monte Carlo

- Loop over *batches*
  - 
  - Loop over *events*
    - •
    - • vector analysis of 1 event for many particle histories
    - ...
  - ...

## Parallel Vector Monte Carlo

- Loop over *batches*
    - Loop over *events*
      - • vector vector ... vector
      - • cpu-1 cpu-2 ... cpu-N
      - ...
      - ...
- parallel ⇒ many particles per CPU  
vector ⇒ events on each CPU
- high-level: parallel  
low-level: vector

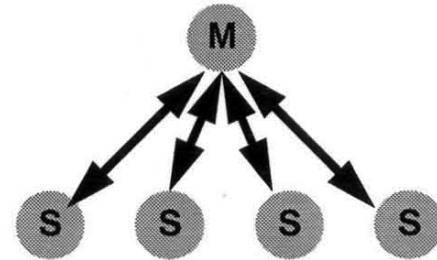
# Hierarchical Parallelism

## RACER parallel algorithm



### High-level:

- independent tasks + **message-passing**
- distribute histories among processors
- Master / Slave algorithm
  - Master: control, distribute work, collect results
  - Slaves: compute particle histories, no communication with other slaves



*(next)*

### Mid-level:

- independent tasks + shared memory
- **"macro-tasking"**
- several slaves share memory, take turns on "critical regions"



### Low-level:

- **"microtasking"**
- split each DO-loop into pieces, compute, synchronize



### Low-level:

- **vectorization**, within each slave process
- Event-based algorithm (Brown/Martin, 1981)
  - vectorize events independently (collision, 3D flight, boundary, ...)
  - create & manage queues of particles waiting for each event

# Parallel MC Algorithms – Alternatives for Shared-Memory

## Shared memory usage

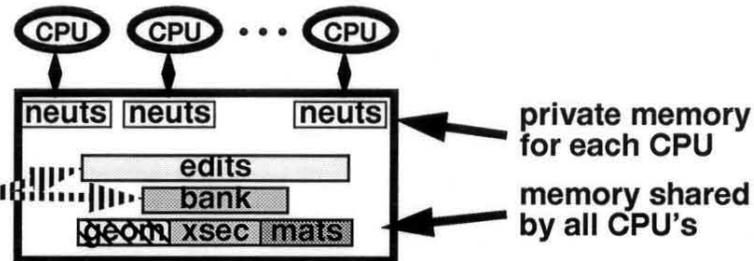
Algorithm decisions: Private data vs. Shared data

Issues: overall memory size, data coherency, memory contention, lock/unlock overhead, portability

### RACER — 1986

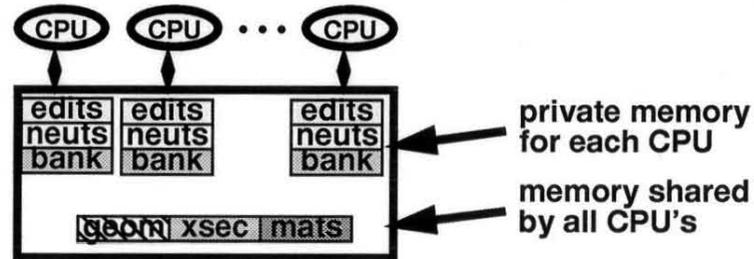
- benchmarks
- Cray-XMP

- LOCK
- mem access
- UNLOCK



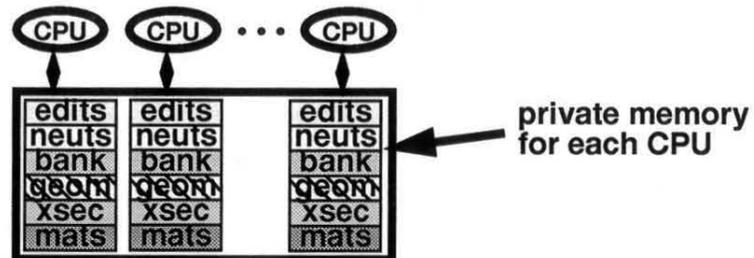
### RACER — 1987

- Cray-XMP



### RACER — 1988...today

- Cray-YMP
- Cray-C90
- super-WS



# Parallel MC Algorithms – Distributed Memory & Clusters

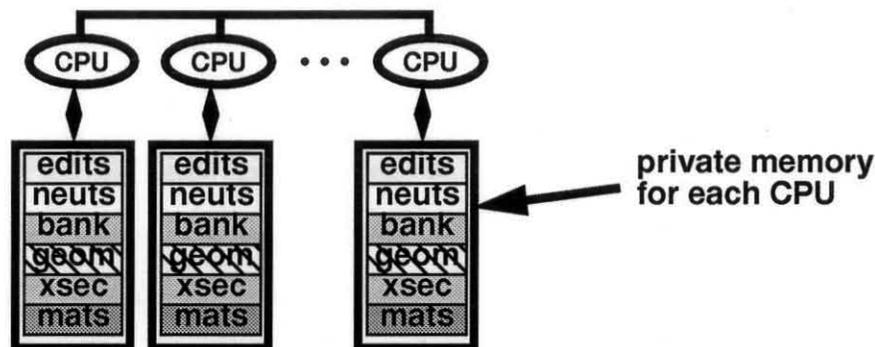
## Distributed memory usage

Algorithm decisions: private data (only)

Issues: local memory size

RACER — 1989...today

- Meiko CS1, 2
- WS cluster



## Clustered Shared-Memory

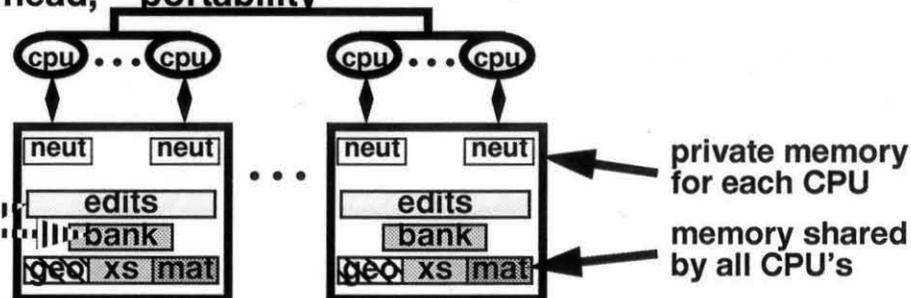
Algorithm decisions: Private data vs. Shared data

Issues: overall memory size, data coherency, memory contention, lock/unlock overhead, portability

RACER — (soon)

- Next super
- Cray-C90
- cluster of super-WS
- cluster of anything

- LOCK
- mem access
- UNLOCK



# Parallel MC – Speedup & Scaling

---

## Performance Measurements

- **Metrics**

$$\begin{array}{ll} \text{Speedup} & S_N = T_1 / T_N \\ \text{Efficiency} & \eta_N = S_N / N \end{array} \quad N = \# \text{ processors}$$

- **Fixed Overall Work**

- Efficiency decreases with **N**
- Speedup (eventually) drops
- Example:
  - constant — # histories/batch
  - variable — # histories/processor ( $\sim 1/N$ )

- **Fixed Work per Processor**

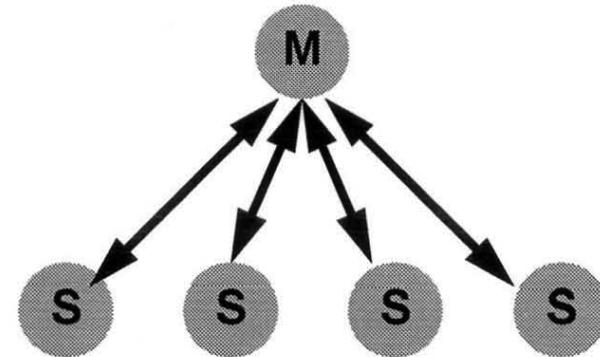
- Efficiency approx. constant with **N**
  - Speedup approx. linear with **N**
  - Example:
    - variable — # histories/batch ( $\sim N$ )
    - constant — # histories/processor
- called "*scaled speedup*"

# Parallel Speedup & Scaling

## Master / Slave Algorithm

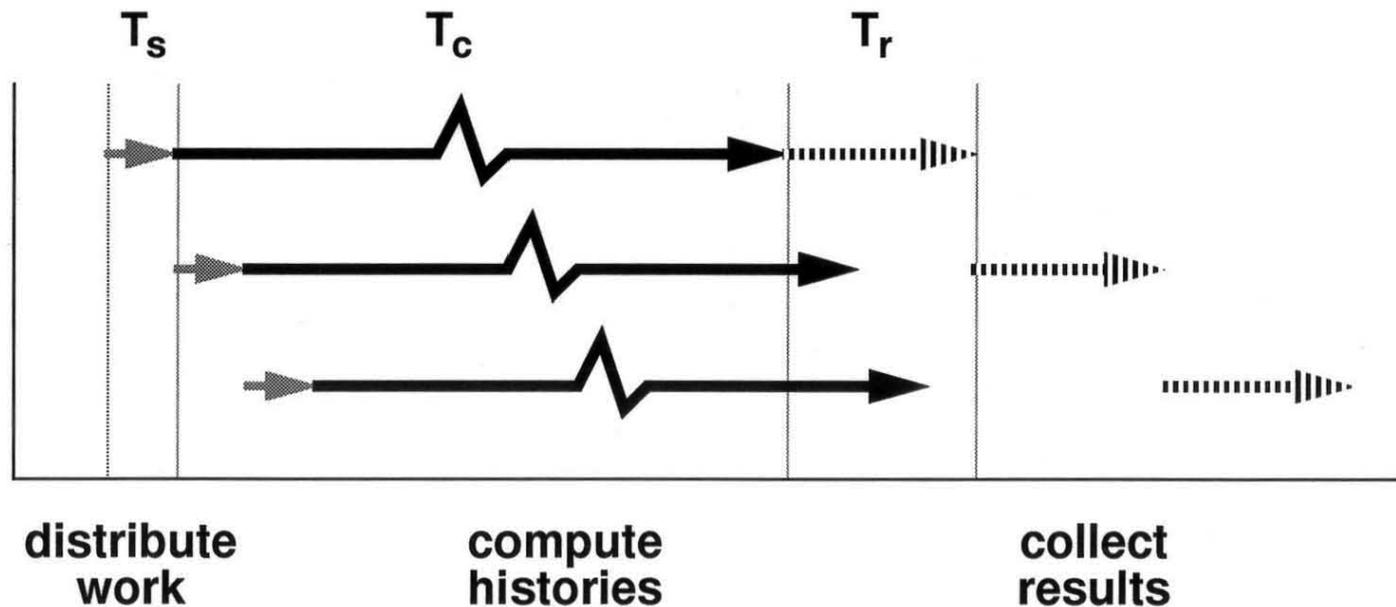
### • Master

- control
- distribute work
- collect results



### • Slaves

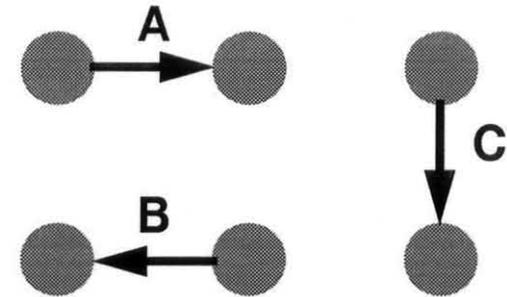
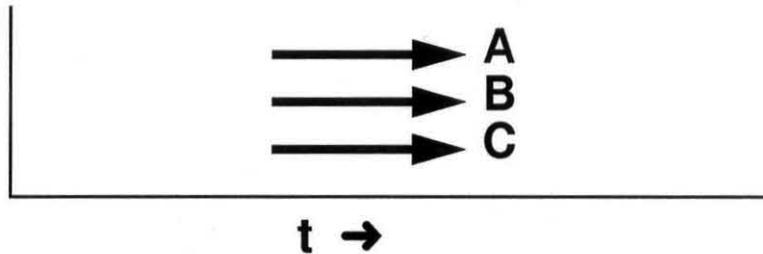
- compute particle histories
- no communication with other **slaves**



# Advanced Computers

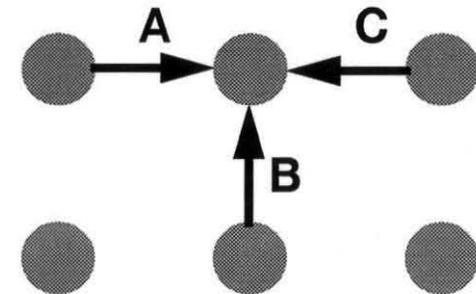
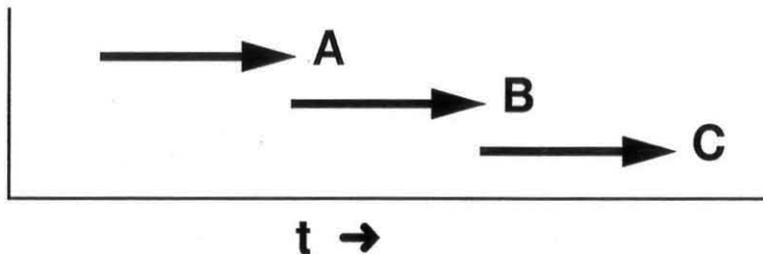
## "Parallel" Message Passing

Most parallel computers support concurrent message-passing between separate pairs of nodes



## "Serial" Message Passing

But, multiple messages to a single node are (almost always) handled sequentially



# Parallel Speedup & Scaling

---

For a given physical problem,

- computation time  $\sim$  number of histories ( $M$ )

• Define:

$M_1$  = number of histories in job (fix.src.) or batch (eig.)  
for calculation using 1 slave

$N$  = number of slave processes

$M_N$  = number of histories per slave  
in job (fix.src) or batch (eig.) using  $N$  slaves

$T_N$  = total time required for  $M_N$  histories =  $T_s + T_c + T_r$

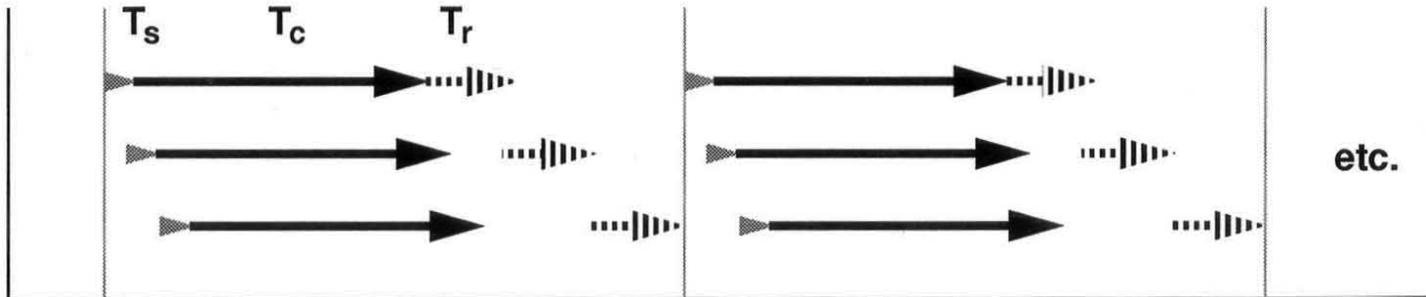
**"Fixed Size" Problem:**  $M_N = M_1 / N$

- Constant number of particles in job (fix.src.) or batch (eig.)
- Goal of parallel calculation: ***same work, less time***

**"Scaled" Problem:**  $M_N = M_1$

- Constant number of particles per slave
- Goal of parallel calculation: ***more work, same time***

# Parallel Speedup & Scaling – Eigenvalue Problems



$t_h$  = **cpu** time per history,

depends on: physics, geometry, Fortran compiler,  
machine speed & architecture,  
computer coding, random straggling, etc.

$L$  = amount of tally data per slave, proportional to # regions

$T_s \approx 0$  negligible — send coordinates to slaves

$T_r \approx s + L / r$   $s$  = latency,  $r$  = streaming rate

$T_c \approx M_N t_h$

$T_c^{\text{fix}} = M_1 t_h / N$  (fixed size)

$T_c^{\text{scale}} = M_1 t_h$  (scaled)

# Parallel Speedup & Scaling

## Scaling Models for Parallel Eigenvalue Calculations

<u>Problem Size</u>	<u>Message Passing</u>	<u>Parallel Speedup</u>
fixed		$S_{\text{fix}} = T_1 / T_N^{\text{fix}}$
"	serial	$S_{\text{fix,ser}} = T_1 / (0 + T_1/N + N T_r) = N / (1 + cN^2)$
"	parallel	$S_{\text{fix,par}} = T_1 / (0 + T_1/N + T_r) = N / (1 + cN)$
scaled		$S_{\text{scale}} = N T_1 / T_N^{\text{scale}}$
"	serial	$S_{\text{scale,ser}} = N T_1 / (0 + T_1 + N T_r) = N / (1 + cN)$
"	parallel	$S_{\text{scale,par}} = N T_1 / (0 + T_1 + T_r) = N / (1 + c)$

$$c = (s + L/r) / (M_1 t_h)$$

# Parallel Speedup & Scaling

## Scaling Models for Parallel Eigenvalue Calculations

problem size

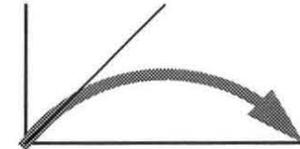
communications

Speedup

fixed

serial

$$S = N / (1 + cN^2)$$



fixed

parallel

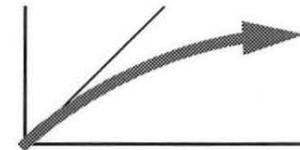
$$S = N / (1 + cN)$$



scaled

serial

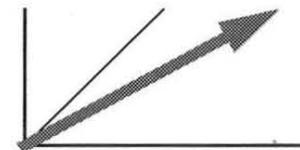
$$S = N / (1 + cN)$$



scaled

parallel

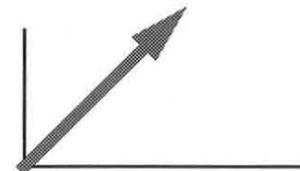
$$S = N / (1 + c)$$



## Scaling Models for Parallel Fixed-source Calculations

for long calculations:

$$S \sim N$$



**N = number of slaves**

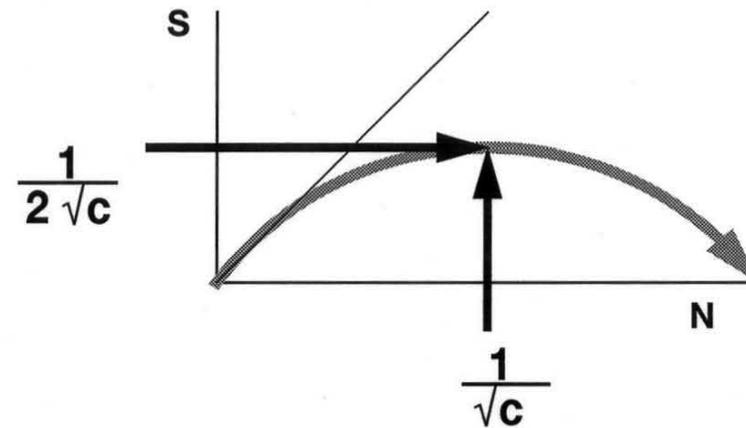
$$c = (s + L/r) / (M_1 t_h)$$

# Scaling - Limits & Metrics

## Parallel Eigenvalue Calculations

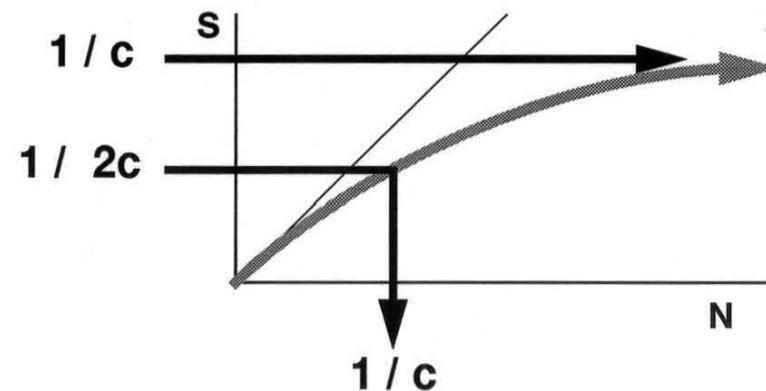
### Fixed size, serial messages

$$S = N / (1 + cN^2)$$



### Scaled size, serial messages

$$S = N / (1 + cN)$$



**N = number of slaves**

$$c = (s + L/r) / (M_1 t_h)$$

# Scaling – Limits & Metrics

## Parallel Eigenvalue Calculations — scaled size, serial messages

$$S = N / (1 + cN)$$

$$S_{\max} = 1 / c$$

$$N_{1/2} = 1 / c$$

$N$  = number of slaves

$$c = (s + L/r) / (M_1 t_h)$$

### Examples:

- **VIM, TREAT problem**

Sun Sparc2 workstation cluster	$c=.0043$	$S_{\max} = 233$
rs6000/350 workstation cluster	$c=.011$	$S_{\max} = 93$
SP1, using ethernet	$c=.014$	$S_{\max} = 70$
SP1, using EUIH comm.	$c=.00134$	$S_{\max} = 748$

- **RACER, "typical" large problem**

100 K histories/min, 20 K histories/slave  
32 MB tally data,  $r \sim 1800$  MB/sec

Cray-C90, using SSD for messages (16 processors, max)	$c \sim .001$	$S_{\max} \sim 1000$
--	---------------	----------------------



# Conclusions

---

- **Parallel Monte Carlo codes ( mcnp4a, keno, vim, racer, ... ) are now running on many parallel computers (cray, meiko, intel, convex, cm5, ...) & workstation networks**
- **Master/slave algorithms are simple, easy to implement, & scale well for eigenvalue problems.**
- **Communications bottlenecks at the master process are not a problem today:**
  - ethernet is fast enough for 10's of slaves
  - FDDI, EUIH, & other schemes should permit 100's of slaves
- **The major limitation on parallel Monte Carlo today appears to be memory size — each node must contain entire problem & tallies**

# Conclusions

---

## Vector & parallel computing have side-benefits:

- Larger batches + more batches →
  - reduces convergence problems
  - reduces bias
  - better correlation corrections
- "Unthinkable" problems become routine

## Proven Monte Carlo algorithms exist for

- SIMD, MIMD, & mixed MIMD/SIMD supercomputers & MP computers
- MIMD distributed processing on a network of machines

# Challenges

---

## Parallel algorithms which scale to 1,000's or 1,000,000's of nodes

- "master-slave" algorithm on 1,000,000 Internet nodes could take ~ 1 year to start
  - UNIX socket connections — limits master to 10's or 100's of nodes
- develop hierarchical parallelism, "clusters-of-clusters"
- parallel histories + geometric decomposition (?)

## Algorithms with load-balancing & fault-tolerance

- "Virtual supercomputer" can change continuously
- recover from lost nodes & hardware failures
- dynamic load balancing
- cooperate with distributed queuing systems

## Acceleration for eigenvalue calculations

- automated procedures for discarding initial batches
- importance sampling or "fission matrix"

## Methods for eliminating bias

## Improved methods for estimating variance, corrections for correlation

**Advanced  
Computational  
Methods for  
Monte Carlo  
Calculations**

# **Optimizing Monte Carlo Calculations**

**Forrest B. Brown**

**National Laboratory Professor, UNM-NE  
Senior R&D Scientist, Monte Carlo, LANL**



# Abstract

---

## Optimizing Monte Carlo Calculations

Forrest Brown, XCP-3, LANL

Improving the performance of a large, complex, production-quality Monte Carlo code is difficult due to the multitude of features and historical constraints. Experienced Monte Carlo code developers recognize that classic optimization techniques applied to code “hot-spots” may result in 20-30% speedups, while very much larger code speedups are possible from improved algorithms.

This talk reviews the initial performance improvements to MCNP6.1 (2013) that were incorporated into MCNP6.1.1 (2014). The improvements included both classic code optimizations and new algorithms. Testing on a variety of problems demonstrated that the performance improvements were effective, yielding speedups by factors of 1.2x - 4x, depending on the type of problem. For criticality problems, speedups were 1.5x - 1.7x.

For many applications, improved algorithms are required to prepare for the new architectures expected from exascale systems in the next 5-10 years. Much more work is planned as part of the MCNP 2020 initiative for improving MCNP6 performance, structure, parallelism, and algorithms.

# Outline

---

- **Introduction**
  - MC darts
  - Monte Carlo & computer history
  - MCNP 2020 & Parallelism
- **Classic Code Optimization**
  - Traditional vs MC codes
  - Performance benchmarks
  - **Classic optimization**
    - Compiler options
    - Strided array ops
    - Inlining & guards
    - Storage allocation
- **Algorithms**
  - Hash-based energy lookup algorithm
  - Sparse storage for the fission matrix
  - Fission bank reordering
  - Random sampling algorithms
  - Parallel Monte Carlo
- **Conclusions**

---

# Introduction

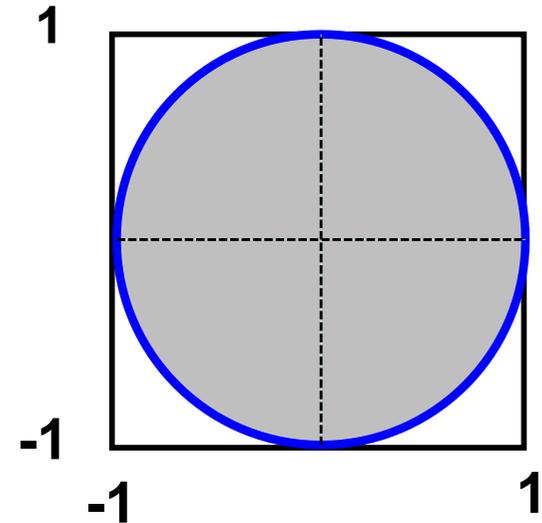
# Monte Carlo Darts Game (1)

- Darts game

Throw darts at a square:

- Sample  $x$  &  $y$  randomly on  $(-1,1)$
- If  $x^2 + y^2 < 1$ , tally a hit

$$\pi \sim 4 * [\# \text{ hits}] / [\# \text{ tries}]$$



- The Monte Carlo "darts" game has been played on some of the biggest and fastest computers around, and has been an informal measure of computer speed. For example,
  - Los Alamos in 1981 stated that **400,000 darts/sec** could be thrown on the Cray-1 computer
  - The challenge to throw darts faster was taken up by F. Brown (KAPL) & W. Martin (Univ. Michigan):
    - **10,000,000 darts/sec** on the Cyber 205 (vector supercomputer)
    - **1 dart/sec** on the HP-11C hand calculator.

# Monte Carlo Darts Game (2)

Year/Place	Machine	Darts / sec
1981 LANL	CDC-7600	0.18 M
1981 LANL	Cray-1	0.40 M
<b>1982 Mich</b>	<b>HP-11C</b>	<b>1</b>
1982 Mich	Apple II+	34
1982 Mich	Amdahl 470V/8	0.17 M
1982 KAPL	Cyber-205, scalar	0.74 M
<b>1982 KAPL</b>	<b>Cyber-205, vector</b>	<b>9.83 M</b>
1999 Mich	233 M PC	0.20 M
1999 Mich	100 M PC	0.07 M
1999 Mich	200 M Pentium, <b>Matlab</b>	446
2002 Mich	900 M P3, <b>Matlab</b>	0.35 M
2002 Mich	900 M P3, <b>Matlab</b> , vec	1.25 M
2002 LANL	1.2 G P3	11 M
2005 LANL	1.0 G P3	19 M
2005 LANL	2.0 G AMD Opteron	24 M
2005 LANL	1.7 G PowerPC G4	32 M
2005 LANL	1.2 G Alpha EV68	101 M
2005 LANL	2.6 G PowerPC G5	140 M

Year/Place	Machine	Darts / sec
2010 LANL	2.6 G i7 2-core, <b>Matlab</b>	0.8 M
2010 LANL	2.6 G i7 2-core	124 M
2010 LANL	2.6 G i7 2-core ***	410 M
2010 LANL	3.0 G 2 Xeon 4-core, 1 thread ***	189 M
2010 LANL	3.0 G 2 Xeon 4-core, 8-thread ***	1460 M
2011 Mich	Linux cluster, MPI, 32 cpu	2000 M
2013 LANL	3.0 G i7 2-core 2-HT	142 M
2013 LANL	3.0 G i7 2-core 2-HT, 1 thread ***	518 M
2013 LANL	3.0 G i7 2-core 2-HT, 2 threads ***	920 M
2013 LANL	3.0 G i7 2-core 2-HT, 4 threads ***	1025 M
2014 LANL	2.4 G 2 i7 4-core, 2-HT, 1 threads ***	194 M
2014 LANL	2.4 G 2 i7 4-core, 2-HT, 8 threads ***	1448 M
2014 LANL	2.4 G 2 i7 4-core, 2-HT, 16 threads ***	2037 M
2014 LANL	2.7 G Xeon 12-core, 2-HT, 12 thrd ***	2670 M
2014 LANL	2.7 G Xeon 12-core, 2-HT, 24 thrd ***	4000 M
<b>2016 LANL</b>	<b>2.7 G Xeon 12-core, 2-HT, 24 thrd ***</b>	<b>5800 M</b>

\*\*\* = hand-tuned, highly optimized

M = MHz, clock speed

G = GHz, clock speed

HT = hyperthreads / core

Fortran, a few Matlab

Note that CPUs, architecture, and compilers all change over time, so that CPU clock speed is not always a good measure of the performance of an application code. This particular comparison is sensitive to 64-bit integer operations (CPU & compiler) and is not necessarily a good predictor of overall Monte Carlo code performance.

# Monte Carlo Darts Game (3)

2.66 GHz Intel Core i7, 64-bit, MacBook Pro (2010)

Straightforward coding

124 M darts/sec

Hand tuned

410 M

2.7 GHz Intel Xeon, 12-core, 2 hyperthreads/core, 64-bit, Mac Pro (2014)

Hand tuned, 24 threads

4000 M

For darts: Mac Pro 2014

~ 10,000x Cray-1

~  $4 \times 10^9$  Bill + HP-11c

~ world pop. + HP-11c



# MCNP 2020

- **MCNP6.1**
  - Preserves old capabilities
  - Many new capabilities
  - RSICC release - July 2013
- **Status**
  - Last few years – focus on features, merger, testing, release
  - Slower, by 30-500 %
- **Path forward – MCNP 2020**
  - Concerted effort to modernize the codebase, upgrade foundations
  - Goals: **faster, sustainable, flexible**
  - Necessary for MCNP to survive into the 2020's & new computers
  - Proposed joint support by DOE-ASC & DOE-NCSP
    - Experienced Lead
    - 2-3 core developers

## MCNP 2020

- **Improve performance**
  - **Goal: 2X speedup within 2 years**
- **Upgrade core MCNP6 software**
  - Restructure, clean up coding, Fortran 2003 & C/C++ standards
  - Reorganize data structures
  - Evolution, not revolution
  - Reduce future costs for new development & maintenance
  - **Goal: sustainable code**
- **Prepare for future**
  - New computers – massive parallel, but less memory per core
  - Improve MPI & thread parallelism
  - **Goal: flexible, adaptable code**

# MCNP 2020 - Performance Improvements

- **Initial 3-month effort, focus on speedup & optimization**
  - Focus on neutron criticality problems common to ASC & NCSP applications
  - **Speedups** from recent performance improvements

## Performance Test Set

### Criticality

ks1 1.76  
 ks2 2.13  
 ks3 1.35  
 ks4 1.36  
 baw1 2.19  
 baw2 1.59  
 fvf 2.04  
 g1 1.14  
 g2 2.20  
 pin 1.73

### Other

void1 3.03  
 void2 4.11  
 void2 4.11  
 void3 2.72  
 det1 1.67  
 med1 1.15  
 pht1 1.22

## VALIDATION\_CRITICALITY Suite

Measured wall-clock times, including data I/O:

mcnp5 release 34.7 min  
 mcnp6.1 release 43.9 min  
 mcnp6.1.1 NEW 27.9 min

- 1.57 X speedup over mcnp6.1
- 1.24 X speedup over mcnp5

## Performance Benchmark Suite

Speedups vs MCNP6.1 Release

### Neutron Problems

BAWXI2 4.37  
 GODIVA 1.05  
 Mode n in air w 750,000 tally bins 1.18  
 Well log problem 1.91  
 100M lattice cells in void 5.17

### Other

mode p e in air 1.01  
 mode n p e in air 1.05  
 mode p in air 1.20  
 Pulse height tally 1.20  
 Radiography 1.07

# Parallel Monte Carlo

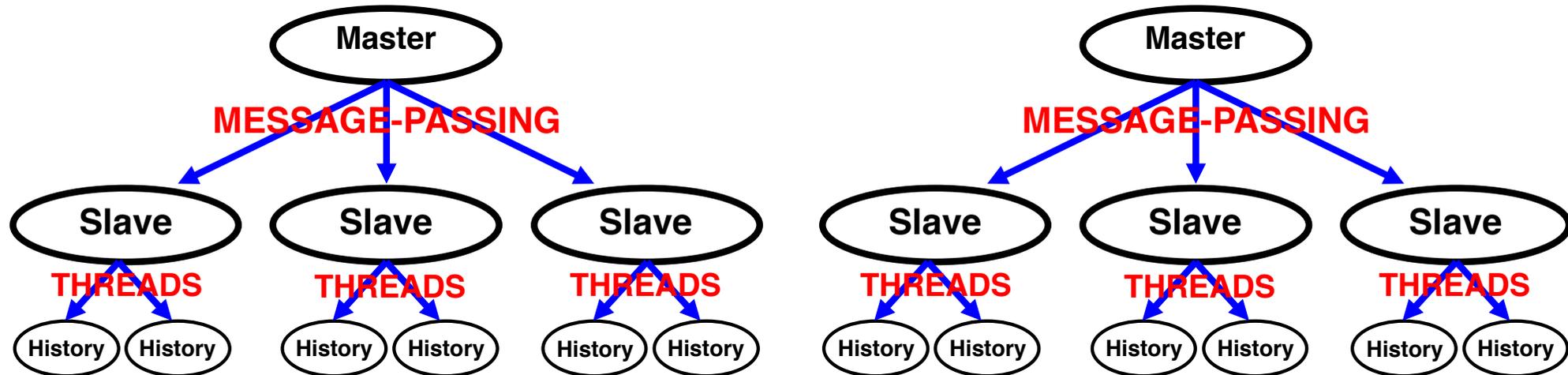
# Computing - Latency & Threading

---

- **Hardware - Moore's Law**
  - Before 2000: 2X cpu speed every 18 months
  - After 2000: more cpu-cores per chip, not faster cpus
  - **Today, hardware speed gains come from parallelism**
- **Fast, multicore cpus**
  - Need more data & need it faster
  - Data transfer speed from memory to CPU has not kept up
  - **Today, data access & latency are biggest concerns**
- **Dealing with latency:**
  - Hardware -- cache, out-of-order execution, multicore, GPUs
  - Algorithms -- High-level, data order & layout, vectorization, threading
  - **Important to match algorithms & hardware**
- **Most large computer systems today are clusters**
  - Many nodes: fiber network interconnect
  - Multicore cpus: share memory within each node
  - **Hierarchical parallelism for Monte Carlo**

# MCNP – Hierarchical Parallelism – Since 2000

Concurrent Jobs →

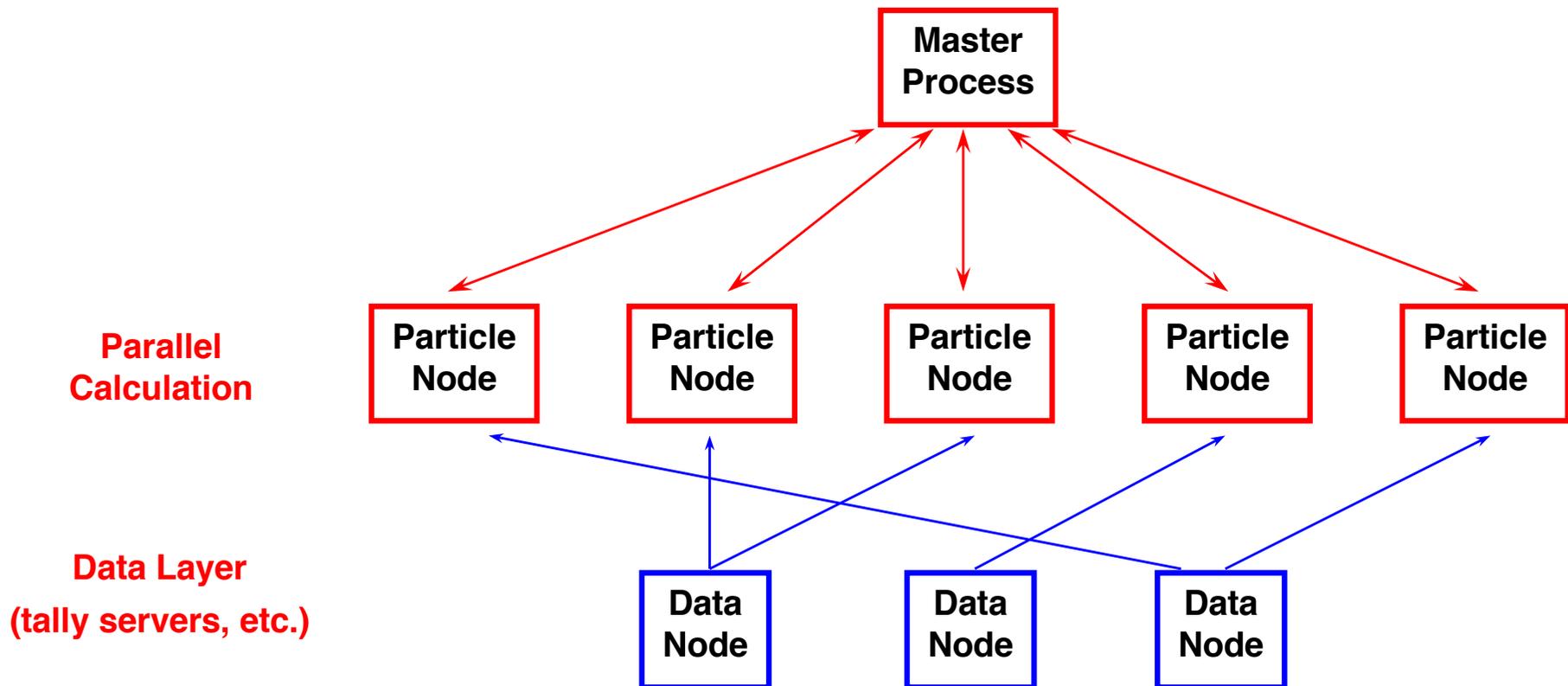


## Parallel Processes

- Total processes =  $(\# \text{ jobs}) \times (\# \text{ MPI processes}) \times (\# \text{ threads})$
- Tradeoffs:
  - More MPI processes - lots more memory & messages
  - More threads - contention from lock/unlock shared memory
  - More jobs - system complexity, combining results

# Parallel Monte Carlo - Future

- Particle parallelism + data decomposition -- logical view:



- Mapping of logical processes onto compute nodes is flexible:
  - Could map particle & data processes to different or same compute nodes
  - Lightweight – particles, heavy-weight – data & tallies
  - Heterogeneous nodes – range of memory, speed, parallelism, etc.

---

# Classic Code Optimization

- **Traditional vs MC codes**
- **Performance benchmarks**
- **Classic Optimization**
  - **Compiler options**
  - **Fix strided array ops**
  - **Inlining & guards**
  - **Storage allocation**

# Traditional vs. MC - Code Optimization (1)

---

- **Traditional:**  
**Use performance tools to find "hot spots" in code execution**
- **Apply classic techniques to optimize coding in hot spots**
  - In-line functions
  - Unroll loops
  - Eliminate unnecessary work (hoist invariants outside loops)
  - Bottom load, top store for loops
  - Vector ops on contiguous data (stride 1)
  - Rearrange storage or loops for contiguous vector ops
  - Etc., etc., etc.
- **For traditional codes (especially mesh-based PDE solvers), focus is typically inner loops in solvers & the floating-point arithmetic**
  - Optimizing data structures & loops can lead to high fractions of overall processor peak speeds

## Traditional vs. MC - Code Optimization (2)

---

- **Breakdown of computer operations for typical large, general-purpose Monte Carlo code (approximate)**
  - 40% - indexing, integer ops, memory access**
  - 30% - test-and-branch**
  - 25% - arithmetic**
  - 5% - RN generation & sampling, 64-bit integers**
- **MC code performance vs. computer hardware**
  - **Memory access is largely random**
    - Little cache-coherency - only small gain from larger cache
    - Memory speed is important
  - **CPU-intensive, but not floating-point**
    - Big gains from multiple integer/logical functional units
    - Smaller gains from multiple floating-point units
  - **Compiler optimizations are critical**
    - Test-and-branch operations, indexing, prefetching
- **MC codes have no hot spots – ops are spread across 100s of routines**
  - **Outer loop over particles, random ops for particles, no inner loops**
  - **Many traditional coding optimization techniques do not apply**

## Traditional vs. MC - Code Optimization (3)

---

- M.C. codes have many levels of indirection for memory access

```
cell
mat = mat_in_cell( cell )
iso = iso_in_mat( i, mat )
k = energy_bin_table_search( E, Eiso(1,iso) )
sig_t = sig_t
        + den*[ (1-de)*sig_t_iso(k,iso) + de*sig_t_iso(k+1,iso) ]
```

- Each level of indirection:
  - Integer ops for indexing
  - Irregular memory access
  - Cache-misses
  - Inhibits pre-fetching, compiler optimization, & vectorization

# Traditional vs. MC - Code Optimization (4)

## Conditionals

- Traditional codes

- Vectorizable loops inside a conditional
- 1 conditional, to skip many ops

```
if( using_some_option ) then
  do k = 1, big_number
    ...vectorizable coding
  enddo
endif
```

- Monte Carlo

- Outer loop over particles, inner coding is scalar (threadable)
- Many conditionals, to skip a few ops
- 1/3 of statements are conditionals, rare options can have significant cost

```
do k = 1, big_number
  if( using_some_option ) then
    ...scalar coding
  endif
enddo
```

## Functions

- Traditional codes

- One function call, vector ops
- Often call, then return if not needed
- Almost no-cost if immediate return

```
call some_option( ...vectors )
```

- Monte Carlo

- Many function calls, scalar ops
- Significant cost to call if not needed

```
do k = 1, big_number
  call some_option( ...scalars )
enddo
```

## Traditional vs. MC - Code Optimization (5)

---

2013-09-11

- **These software practices are bad inside Monte Carlo histories:**
  - **Assuming that the compiler will inline functions**
    - No inlining is done for the safe optimization level used for mcnp
  - **Using accessor functions to determine if an option is in effect**
    - Requires an external call, invoked very many times
  - **Calling an unneeded routine, even if it exits immediately**
    - Requires an external call, invoked very many times
  - **Eliminating goto statements by pushing coding into a subroutine**
    - Requires an external call, invoked very many times
  - **Adding extra levels of looping just to avoid goto's for very rare cases**
    - Extra overhead on all particles; less understandable code
  - **Heavy use of loop constructs cycle & exit is as bad as goto's**
    - Obscures code flow & logic
  - **Obsession with removing goto's**
    - They have their place in MC, more so than in other types of algorithm

## Traditional vs. MC - Code Optimization (6)

---

### What classic optimizations work for MC ?

- **Eliminate unneeded work, wherever possible**
- **Replace any code constructs that require temporary storage (eg, noncontiguous array ops, character manipulation, ...)**
- **Replace calls to accessor functions by direct inline access**
- **Put if-tests for options inline, not in external routines**
- **Put (short) functions inline, not in external modules**

# Performance Benchmarks

2013-09-11

All tests run on Mac Pro, 3.0 GHz Xeon, 2 quad-cores using 8 threads. For criticality problems results are neutrons/hr; for fixed-source problems results total wall time. ENDF/B-VII.1; Only discrete S(a,b) was used.

## CRITICALITY PROBLEMS

ks1.txt 3D PWR, OECD perf. bench., Kord Smith, **60 isotopes, no tallies**

ks2.txt ks1.txt, **10 isotopes, no tallies**

ks3.txt ks1.txt, **10 isotopes, fmesh tallies**

ks4.txt ks1.txt, **60 isotopes, fmesh tallies**

baw1.txt BAWXI2 ICSBEP problem, **31 isotopes, no tallies**

baw2.txt BAWXI2 ICSBEP problem, **31 isotopes, fmesh tallies**

fvf.txt fuel storage vault, from OECD convergence benchmarks

g1.txt Godiva problem, **3 isotopes**

g2.txt Godiva problem, **423 isotopes**

pin.txt AECL pin cell, with FPs, **147 isotopes**

## FIXED-SOURCE PROBLEMS

void1.txt ks1.txt, with VOID card & no tallies

void2.txt baw1.txt, with VOID card & no tallies

void3.txt fvf.txt, with VOID card & no tallies

det1.txt 3D porosity tool, Reg. problem 12, neutrons, weight windows, F4 tallies

med1.txta medical physics, modified 3D Zubal head, **photons**

pht1.txt PHTVR cylindrical test problem, **photons**

# Classic Optimization - Compiler Options (1)

2013-07-24

## Try different compiler optimization levels

### Test case:

- BAWXI2 criticality benchmark, endf-7.0, 250 cycles, 5K neutrs/cycle
- Mac OS X, Intel-12, 8 threads

### Results:

compile options	neutrons/hr	relative speed
<b>MCNP5, RSICC version</b>		
-O1	86 M	<b>1.0</b>
<b>MCNP6.1, RSICC version</b>		
-O1	58 M	<b>.67</b>
-O2	57 M	.66
-O3	57 M	.66

→ No gains from higher compiler optimization level (-O1, -O2, -O3)  
(Some other test problems segfault for -O2, -O3)

# Classic Optimization - Compiler Options (2)

2013-07-24

Try different **heap-array allocation** for temporary storage

## Test case:

- BAWXI2 criticality benchmark, endf-7.0, 250 cycles, 5K neutrs/cycle
- Mac OS X, Intel-12, 8 threads

## Results:

compile options	neutrons/hr	relative speed
<b>MCNP5, RSICC version</b>		
<b>-O1 -heap-arrays 1024</b>	<b>86 M</b>	<b>1.0</b>
<b>MCNP6.1, RSICC version</b>		
<b>-O1 -heap-arrays 1024</b>	<b>37 M</b>	<b>.43</b>
<b>-O1 -heap-arrays 16384</b>	<b>39 M</b>	<b>.46</b>
<b>-O1 -heap-arrays 1048576</b>	<b>38 M</b>	<b>.45</b>

→ **No gains from larger heap-array allocation**

(Some other test problems segfault if heap-array allocation not used)

# Classic Optimization – Inlining Functions

## Modifications to original mcnp6.1:

- **Inline binary searches** in neutron problems for cross-section data, tallies, etc.
- **Eliminate unnecessary calls to external routines**, using extra logical variables for global options
- **Inline external routines** for neutron problems, ~10 routines in collision physics

→ Roughly 5-15% gain in overall code speed due to moderate inlining

speedup  
due to inlining

### KCODE

ks1	1.34
ks2	1.16
ks3	1.11
ks4	1.11
baw1	1.09
baw2	1.07
fvf	1.09
g1	1.00
g2	1.14
pin	1.05

### FIXED-SOURCE

void1	1.04
void2	1.05
void3	0.96
det1	1.08
med1	0.99
pht1	1.05

All problems run on Mac Pro (3 GHz Xeon) with 8 threads, Intel 12.0

# Classic Optimization – Storage Allocation

## Fortran Common Blocks & Threading Performance

- In MCNP6.1, each thread-private variable used in particle tracking was **individually & explicitly declared to be THREADPRIVATE**. COMMON blocks were not used.
  - In MCNP6.1.1, all thread-private variables used in particle tracking were placed in COMMON blocks & **only the COMMON block names are declared THREADPRIVATE**
- Roughly 5-20% gain in overall code speed due to changes in thread-private declaration
- Very compiler-dependent
  - Apparently more addressing ops needed when each variable declared separately

speedup due to  
thread-private  
COMMON

### KCODE

ks1.txt	1.12
ks2.txt	1.17
ks3.txt	1.12
ks4.txt	1.06
baw1.txt	1.28
baw2.txt	1.18
fvf.txt	1.31
g1.txt	1.09
g2.txt	1.02
pin.txt	1.07

### FIXED-SOURCE

void1.txt	1.08
void2.txt	1.03
void3.txt	1.08
det1.txt	1.24
med1.txt	1.00
pht1.txt	1.12

All problems run on Mac Pro (3 GHz Xeon) with 8 threads, Intel 12.0

# Classic Optimization – Combined Gains

- Overall speedups due to recent coding optimization

**Modifications:**

- Compiler options
- Fix strided array ops
- Inlining & guards
- thread-private common

- Comments

- Focus for classic optimizations was neutron criticality problems
- Classic optimizations focused on coding, not algorithms
- Many more improvements could be made
- Effort only required ~2 months, hardest part was testing on a variety of problems

**Overall speedup  
vs mcnp6.1**

**KCODE**

ks1	1.44
ks2	2.11
ks3	1.34
ks4	1.23
baw1	1.87
baw2	1.43
fvf	1.97
g1	1.17
g2	1.19
pin	1.19

**FIXED-SOURCE**

void1	2.96
void2	4.02
void3	2.71
det1	1.59
med1	1.07
pht1	1.21

All problems run on Mac Pro (3 GHz Xeon) with 8 threads, Intel 12.0

---

# Algorithms

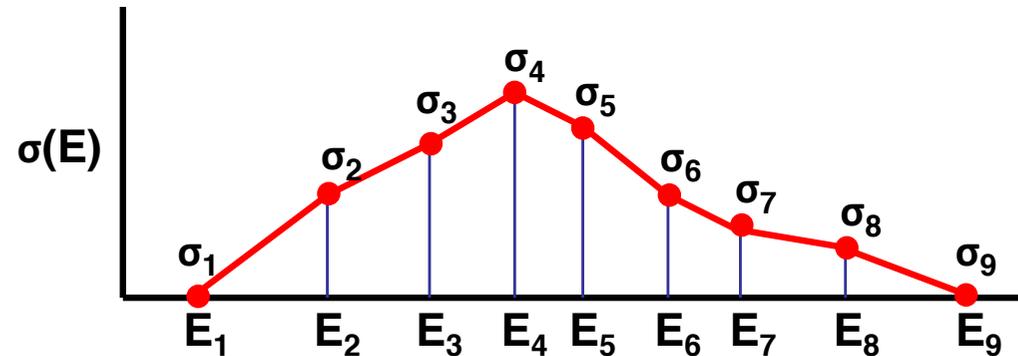
- **Coding optimizations are easy, but provide only limited speedups.**
- **Speedups from code optimization are often compiler-dependent & need to be revisited when new compilers are used.**
- **The biggest gains always come from new algorithms.**
- **New algorithms are needed for the coming new computer architectures: cpu + mic + gpu, billions of cores, limited memory**

# Hash-based Energy Lookup Algorithm

# Hash-based Energy Lookup – Background (1)

- Cross-section data are stored as **piecewise linear functions of E**

– Typical  $\sigma(E)$  vs E



- Usually stored as linear arrays:

N = number of entries

$E(1..N)$  = array of E values =  $(E_1, E_2, \dots, E_N)$

$\sigma(1..N)$  = array of  $\sigma$  values =  $(\sigma_1, \sigma_2, \dots, \sigma_N)$

- Two steps are required to lookup & use the data:

- Given E, **search** the E() array to find interval k containing E ( $1 \leq k \leq N-1$ )
- Interpolate** linearly between  $E_k$  &  $E_{k+1}$

$$\sigma(E) = \sigma_k + \left( \frac{E - E_k}{E_{k+1} - E_k} \right) \cdot (\sigma_{k+1} - \sigma_k), \quad E_k \leq E \leq E_{k+1}$$

## Hash-based Energy Lookup – Background (2)

---

- **After a collision (before a flight) or entering new material**
  - Must look up & interpolate  $\sigma_T$  for the neutron energy  $E$ , for each nuclide in a material
  - The  $\sigma_T$ 's are used to determine  $\Sigma_T$  for the material
  - $\Sigma_T$  is then used in randomly sampling of distance to collision

For U235, U238, O16, ... (fuel material)

- . Search the array of energies for the nuclide, find interval  $k$  containing  $E$
- . Interpolate  $\sigma_T$  for nuclide at energy  $E$
- . Accumulate  $N\sigma_T$  for nuclide into overall material  $\Sigma_T$
- ...

Similar {search, interpolate, accumulate} for scatter, absorption, fission, ...

- **This set of operations {search, interpolate, accumulate} often consumes 1/3 – 2/3 of the overall time in neutron transport MC**

## Hash-based Energy Lookup – Background (3)

---

- **There is extensive literature on search algorithms**
  - D.E. Knuth, The Art of Computer Programming Vol 3 - Sorting & Searching
  - Many other references - books & journals
- **For general Monte Carlo codes, the commonly-used methods are linear search &/or binary search of the cross-section energy tables**
  - Need 1 table search for each of the nuclides in a material
  - Linear search takes  $O(N)$  time, best when  $N \sim 10$  or less
  - Binary search takes  $O(\ln N)$  time, best when  $N \sim$  large
- **To reduce the time needed for the table searches for cross-section data, several unified energy grid schemes were used in the past**
  - Map the data for every nuclide in the problem onto 1 energy grid
  - Requires only 1 energy table search, rather than 1 table search for every nuclide in a material
  - Can be 10-100x faster for energy lookups

# Hash-based Energy Lookup – Background (4)

---

## Unified Energy Grid Schemes

- **Scheme 1 – very old** (racer, rcp, o5r, ...)
  - Used in the 1960s – 1980s due to memory limitations
  - Typically  $10^4$  –  $10^5$  energy bins (supergrouped)
  - Map all xsec data to these bins
  - Approximate, required weighting functions
- **Scheme 2 – unified grid** (psg, serpent, ...)
  - Combine all xsec energy grids, including all energy points
  - Expand all xsec data onto unified grid
  - Exact, but required very large amounts of memory
- **Scheme 3 – unified grid with pointers** (serpent, ...)
  - Combine all xsec energy grids, including all energy points
  - For each unified grid bin, store pointers to bins in each nuclide xsec data set
  - Exact, retains original nuclide xsec data
  - Extra storage for unified grid & nuclide pointers
  - Requires only 1 table search, then (indirect) lookups in nuclide tables
- **Scheme 4 – NEW, current hash-based energy lookup**  
(mcnp611)

# Hash-based Energy Lookup – Background (5)

## Unified Energy Grid Schemes – Memory Storage

	<u>nuclides</u>	<u>E pts</u>	<u>ACE xs</u>	<u>Ugrid+xs</u>	<u>Ugrid+ptrs</u>	<u>NEW</u>
<b>K Smith bench</b>	64	.73 M	.12 GB	1.5 GB	.38 GB	2.2 MB
<b>Rx pin, 1 temp</b>	77	.66 M	.12 GB	1.6 GB	.41 GB	2.6 MB
<b>Rx pin, 2 temps</b>	145	1.2 M	.24 GB	5.6 GB	1.4 GB	4.8 MB
<b>Rx pin, 5 temps</b>	349	2.8 M	.55 GB	31 GB	7.8 GB	12 MB
<b>All nucs, 1 temp</b>	423	2.6 M	.58 GB	36 GB	9.0 GB	14 MB

**ACE xs** = actual memory for ACE data in mcnp611

**E pts** = total energy points, summed over all ACE nucs = pts in Ugrid

**Ugrid+xs** = extra storage for unified E-grid +  $\{\sigma_T, \sigma_A, \sigma_E, \text{heating}\}$  at each E & nuc

**Ugrid+ptrs** = extra storage for unified E-grid + pointers to nuc xsecs at each E & nuc

**NEW** = extra storage for current hash-based lookup, with 8192 ubins

# Hash-based Energy Lookup – New (1)

---

- **History**

- Suggested by George Zimmerman (LLNL, ret.) in 2013
- Used for lattice physics code (Dave Austin) in ~1989, and in several variations in RACER MC (Brown) in 1980s
- Certainly much older .....

- **Recent**

- Zimmerman, in proprietary code mods, 2013
- Brown, stand-alone & in mcnp6.1.1b, 2013-2014

- **Basic idea**

Retain all mcnp6 machinery for energy lookups & forming the total cross-section, but

- use a physics-based hash scheme to greatly narrow the bounds for each binary search of nuclide E tables
- Minimal mcnp6 code changes, but significant speedups
- Modest memory storage, much less than unified grids

## Hash-based Energy Lookup – New (2)

---

- The setup portion of the algorithm, performed prior to neutron random walks, involves the following steps:
  1. Determine  $E_{min}$  and  $E_{max}$  energy bounds for the problem
    - Check  $E_{max}$  &  $E_{min}$  for all nuclide ACE datasets in the problem
  2. Setup the "ugrid" for the hashing function
    - **Ugrid**: uniform spacing in  $\ln(E)$  between  $E_{min}$  and  $E_{max}$
    - **M**: number of bins in  $ugrid()$ .
    - No need to store  $ugrid()$  -- just store  $M, E_{min}, E_{max}$
    - mcnp611:  $M = 8192$ , reasonable speed/storage tradeoff
  3. Setup nuclide search bounds for each ubin index
    - For each bin in  $ugrid$ , lookup & store for each nuclide the bounding indexes  $k_1(u,n)$  and  $k_2(u,n)$  in the ACE energy table for that nuclide ( $n$ = nuclide index,  $N$ = no. nuclides,  $u$ = index in  $ugrid$  )
    - Only need store  $k_1(u,n)$ , since  $k_2(u,n) = k_1(u+1,n)+1$
    - Total extra storage =  $(M+1) \cdot N \cdot 4$  bytes (int4 sufficient for ACE data)

**Note:** The above steps do NOT involve any approximations

## Hash-based Energy Lookup – New (3)

- During random walk simulation, after particle energy change or when entering new material

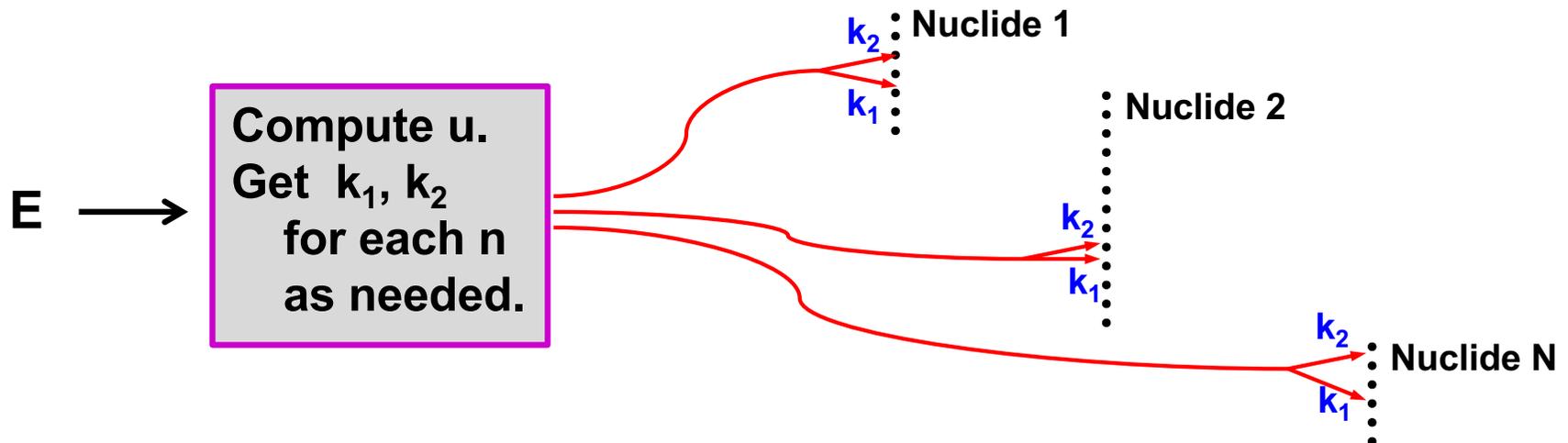
Defining  $u_{min} = \log E_{min}$ ,  $u_{max} = \log E_{max}$ ,  $du = M / (u_{max} - u_{min})$

New algorithm for energy lookups for neutron energy  $E$  is:

$$u = 1 + \lfloor du \cdot (\log E - u_{min}) \rfloor, \quad \lfloor \rfloor \text{ is truncation to the next lowest integer}$$

For each nuclide  $n$ :

search its energy table between entries  $k_1(u, n)$  &  $k_2 = k_1(u+1, n) + 1$



## Hash-based Energy Lookup – New (4)

---

- **Memory storage**
  - *ugrid* is completely defined by  $M, E_{min}, E_{max}$  -- need not be stored
  - Because  $k_2(u,n) = k_1(u+1,n)+1$ , the  $k_2(u,n)$  values need not be stored
  - Total additional memory storage =  $(M+1) \cdot N \cdot 4$  bytes
  - More compact memory use, so more cache-friendly
- **Speed/space tradeoff**
  - Larger  $M$  gives improved speed, but dependence is weak for  $M > 1000$
  - Smaller  $M$  reduces speedup but also reduces memory requirements.
- **Choice of  $M$  does not in any way affect accuracy of the xsec data**
- **$k_1$  and  $k_2$  indexes for each nuclide for each of the *ugrid* bins**
  - Bounds for performing ordinary binary searches in the nuclide ACE
  - These bounds narrow the range of the binary searches, so that only a small portion of each nuclide energy table need be searched
  - Frequently the search range in the nuclide energy tables is  $< 8$ .  
For such small ranges, a simple linear search will be slightly faster than a binary search & may provide additional small speedups

# Hash-based Energy Lookup – Testing (1)

---

- **Stand-alone coding to compare 3 methods:**
  1. Standard MCNP6.1 with external function for binary searches
  2. Standard MCNP5 with inline coding for binary searches
  3. New hash-based scheme with inline binary searches.
- **ACE datasets**
  - The energy tables for 9 nuclides from the ENDF/B-VII.1 nuclear data libraries were used in the comparisons:  
1001.80c, 8016.80c, 26056.80c, 92235.80c,  
92238.80c, 94239.80c, 94240.80c, 94241.80c, 6000.80c.
  - These nuclides had energy table sizes ranging from 590 to 157,744 bins.
- For each energy lookup scheme, many millions of neutron energies were randomly sampled in the *ugrid* range, and then the energy lookups were performed for all 9 nuclides. Overall timing results are averages for the set of nuclides.

## Hash-based Energy Lookup – Testing (2)

- Timing results for **stand-alone** test of energy lookup methods. Results are the average time for each energy lookup

	Mac Pro, 3 GHz Xeon, 667 MHz mem	MacBook, 3 GHz i7, 1600 MHz mem
MCNP6.1 energy lookup, with external binary search function	<b>97 ns</b>	<b>67 ns</b>
MCNP5 energy lookup, with explicit inline binary search coding	<b>81 ns</b>	<b>57 ns</b>
New hash-based energy lookup, with explicit inline binary search coding	<b>6 ns</b>	<b>3 ns</b>

- Inlining binary searches gives 10-20% speedup (mcnp5 vs mcnp6.1)
- New hash-based scheme gives 15-20x speedup
- M = 8192 used for table
- Lookup time for other M on MacBook
 

M = 64 k	2 ns
M = 32 k	2 ns
M = 16 k	2 ns
M = 8 k	3 ns
M = 4 k	3 ns
M = 1 k	5 ns
- Mixed binary/linear search (break at 8) did not improve speedup

## Hash-based Energy Lookup – Testing (3)

---

- **MCNP6.1 (2013) runs significantly slower than MCNP5**
  - Slowdowns are problem-dependent, 20% to 5x slower
- **MCNP6.1.1 (2014)**
  - **Significant classic optimizations performed**
    - Inline functions, eliminate non-unit-stride vector ops, if-guards, ...
  - **New hash-based energy lookup scheme**
    - **Measured timing results for new energy lookup scheme compare mcnp6.1.1 before & after new scheme, with all other optimizations the same**
- **New energy lookup scheme provides 1 – 1.9x speedup in overall MCNP6.1.1 problem runtime (at least for neutron problems)**
- **MCNP6.1.1 is a lot faster than MCNP6.1**
- **MCNP6.1.1 is a little faster than MCNP5**

## Hash-based Energy Lookup – Testing (4)

- **MCNP6.1.1 speedups due to new hash-based energy lookup algorithm**

Problem	Overall Code Speedup
OECD Performance Benchmark, 3D PWR, 60 isotopes, no tallies	<b>1.2x</b>
BAW XI(2), ICSBEP Problem LEU-COMP-THERM-008, Case-2, 31 isotopes, no tallies	<b>1.2x</b>
Godiva problem, 2 isotopes, no tallies	<b>1.0x</b>
Godiva problem, with trace amounts of 421 other isotopes	<b>1.9x</b>
Reactor pin cell with 147 isotopes	<b>1.5x</b>
Porosity tool for well-logging, 5 isotopes	<b>1.0x</b>

- Speedup compares mcnp6.1.1 before & after new energy lookup scheme, with no other changes
- M = 8192 used for table
- All runs performed on Mac Pro (3 GHz, 2 quad-core) with 8 mcnp6 threads, using standard ENDF/B-VII data

# **Sparse Storage for Fission Matrix Tallies**

# Fission Matrix for MCNP

- Exact integral equation for fission source**

$$F_{I,J} = \int_{\vec{r} \in V_I} d\vec{r} \int_{\vec{r}_0 \in V_J} d\vec{r}_0 \frac{S(\vec{r}_0)}{S_J} \cdot \int \int \int dE d\hat{\Omega} dE_0 d\hat{\Omega}_0 \cdot v \Sigma_F(\vec{r}, E) \cdot \frac{\chi(E_0)}{4\pi} \cdot G(\vec{r}_0, E_0, \hat{\Omega}_0 \rightarrow \vec{r}, E, \hat{\Omega})$$

$$S_J = \int_{\vec{r}' \in V_J} S(\vec{r}') d\vec{r}' = \int \int \int_{\vec{r}' \in V_J} d\vec{r}' dE' d\hat{\Omega}' v \Sigma_F(\vec{r}', E') \Psi(\vec{r}', E', \hat{\Omega}')$$

$$S_I = \frac{1}{K} \cdot \sum_{J=1}^N F_{I,J} \cdot S_J \quad \mathbf{N = \# \text{ spatial regions, } F \text{ is } N \times N \text{ matrix}}$$

- $F_{I,J}$  = next-generation fission neutrons produced in region I, for each fission neutron starting in region J (J→I)**
  - As region size decreases:  $S(r_0) \rightarrow S_J / V_J$ , discretization errors  $\rightarrow 0$
  - Can accumulate tallies of  $F_{I,J}$  even if not converged

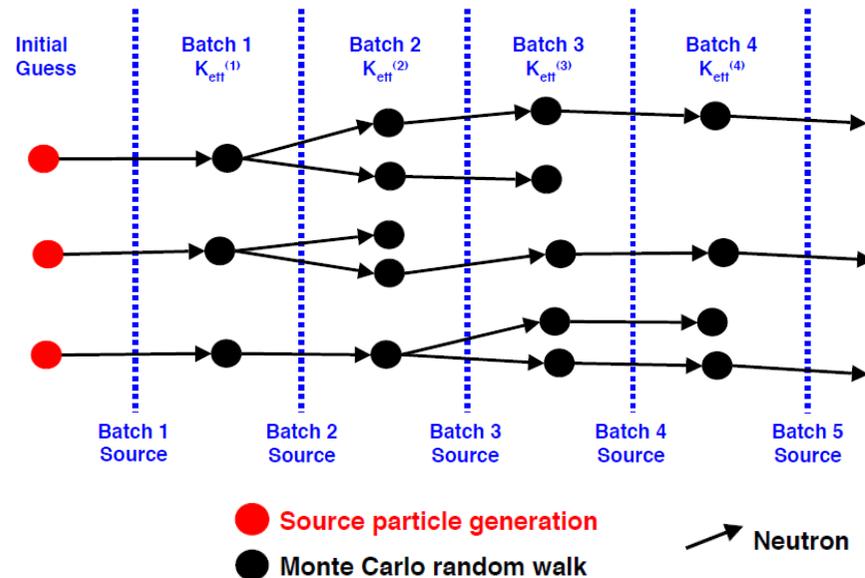
- Similar analysis for adjoint source shows that**

$$S_I^\dagger = \frac{1}{K} \cdot \sum_{J=1}^N F_{I,J}^\dagger \cdot S_J^\dagger, \quad \bar{F}^\dagger = \bar{F}^T$$

# Monte Carlo Estimation of Fission Matrix

## Monte Carlo K-effective Calculation

1. Start with fission source & k-eff guess
2. Repeat until converged:
  - Simulate neutrons in cycle
  - Save fission sites for next cycle
  - Calculate k-eff, renormalize source
3. Continue iterating & tally results



## For Fission Matrix calculation

During standard k-eff calculation, at the end of each cycle:

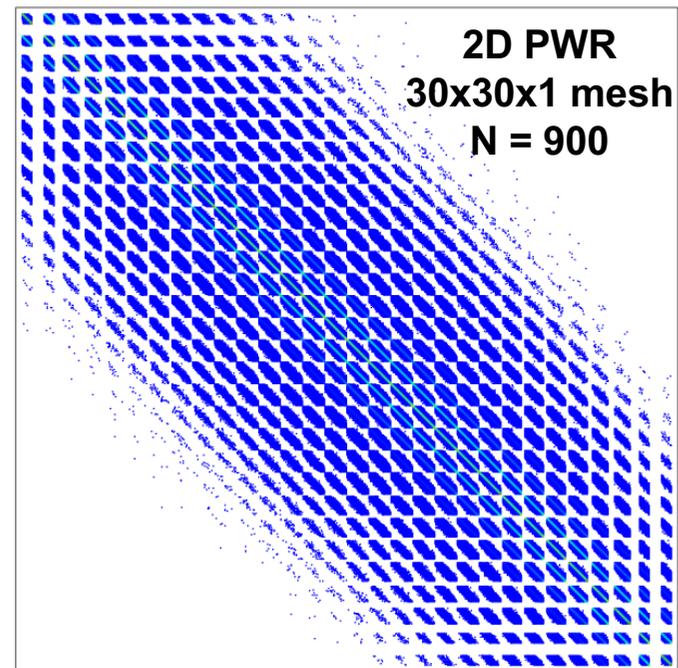
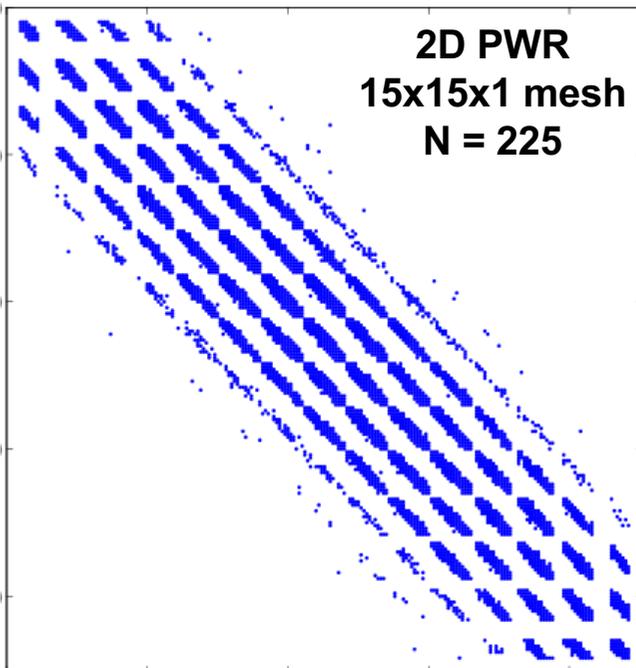
- Estimate  $F_{i,j}$  tallies from start & end points in fission bank ( $\sim$  free)
- Accumulate  $F_{i,j}$  tallies, over all cycles (even inactive cycles)

After Monte Carlo completed:

- Normalize  $F_{i,j}$  accumulators, divide by total sources in J regions
- Find eigenvalues/vectors of F matrix (nonsymmetric eigensolver)

# Fission Matrix – Sparse Structure

- For a spatial mesh with  $N$  regions,  $F$  matrix is  $N \times N$ 
  - 100 x 100 x 100 mesh  $\rightarrow F$  is  $10^6 \times 10^6 \rightarrow$  8 TB memory
  - In the past, memory storage was always the major limitation for  $F$  matrix
- **Compressed row storage scheme**
  - Don't store zero elements, general sparsity
  - Reduced  $F$  matrix storage, no approximation
  - Can easily do 100 x 100 x 100 mesh on 8 GB Mac



# Fission Matrix – Sparse Storage

## • Compressed Row Storage Scheme (CRS)

– General sparsity, no approximations or assumptions

–  $N = N_x \times N_y \times N_z$  mesh cells

–  $(i_S, j_S, k_S) \rightarrow (i_T, j_T, k_T) \rightarrow J \rightarrow I$

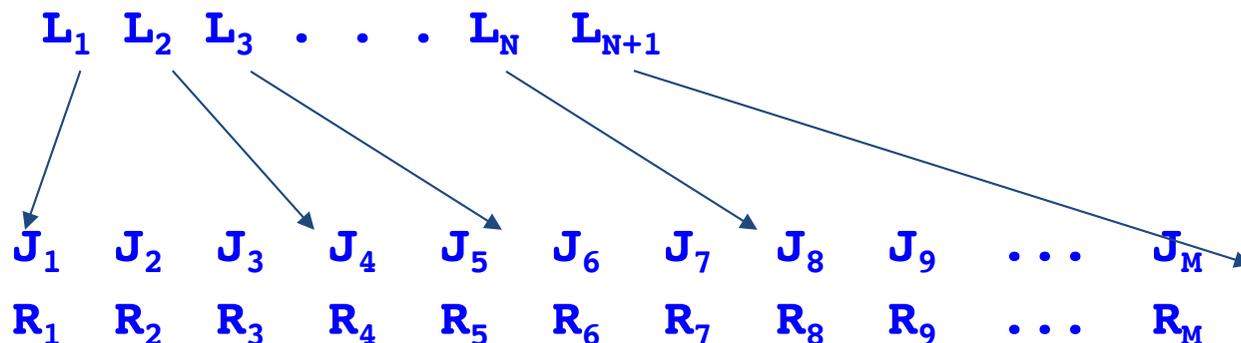
$$J = i_S + (j_S - 1)N_x + (k_S - 1)N_x N_y$$

$$I = i_T + (j_T - 1)N_x + (k_T - 1)N_x N_y$$

– Only the nonzero  $F(I, J)$  entries are stored.

– **MC tallies:** If element exists – add; if not – insert

–  $L(I)$  array entries point to the start of a list of  $J$  indices and corresponding nonzero  $F(I, J)$  tallies



– **Highly optimized tally coding, typically requires less than 1 second at the end of each batch in the Monte Carlo simulation.**

## Example – Sparse-Matrix \* Vector

---

```

! multiply a fmat matrix times a vector, return result in y vector
type(fission_matrix), intent(in)  :: fmat    ! sparse fission matrix
real(R8),              intent(in)  :: x(:)   ! vector in
real(R8),              intent(out) :: y(:)   ! vector out, result
integer(I8) :: k, i
real(R8)     :: t

!$OMP PARALLEL DO PRIVATE( t, k )           ← different thread for each row
do i = 1, fmat%n
  t = 0.0d+00
  do k = fmat%L(i), fmat%L(i+1)-1         ← k is location of J,R row data
    t = t + fmat%R(k) * x(fmat%J(k))
  enddo
  y(i) = t
enddo
!$OMP END PARALLEL DO

```

# Higher Eigenmode Analysis with the Fission Matrix

---

- **Run Monte Carlo, get fission matrix, then solve for eigenvalues & eigenfunctions:**
  - Matlab, if full-storage F matrix can fit in memory
  - Power iteration with deflation, preserves sparse format
  - Implicitly Restarted Arnoldi Method (IRAM), preserves sparse format (thanks, Max & Colin)

$$\vec{S}_n = \frac{1}{K_n} \cdot \bar{F} \cdot \vec{S}_n \quad k_0 > |k_1| > |k_1| \dots > |k_N|$$

$$\vec{S}_n^\dagger = \frac{1}{K_n} \cdot \bar{F}^T \cdot \vec{S}_n^\dagger \quad n = 0, 1, \dots, N$$

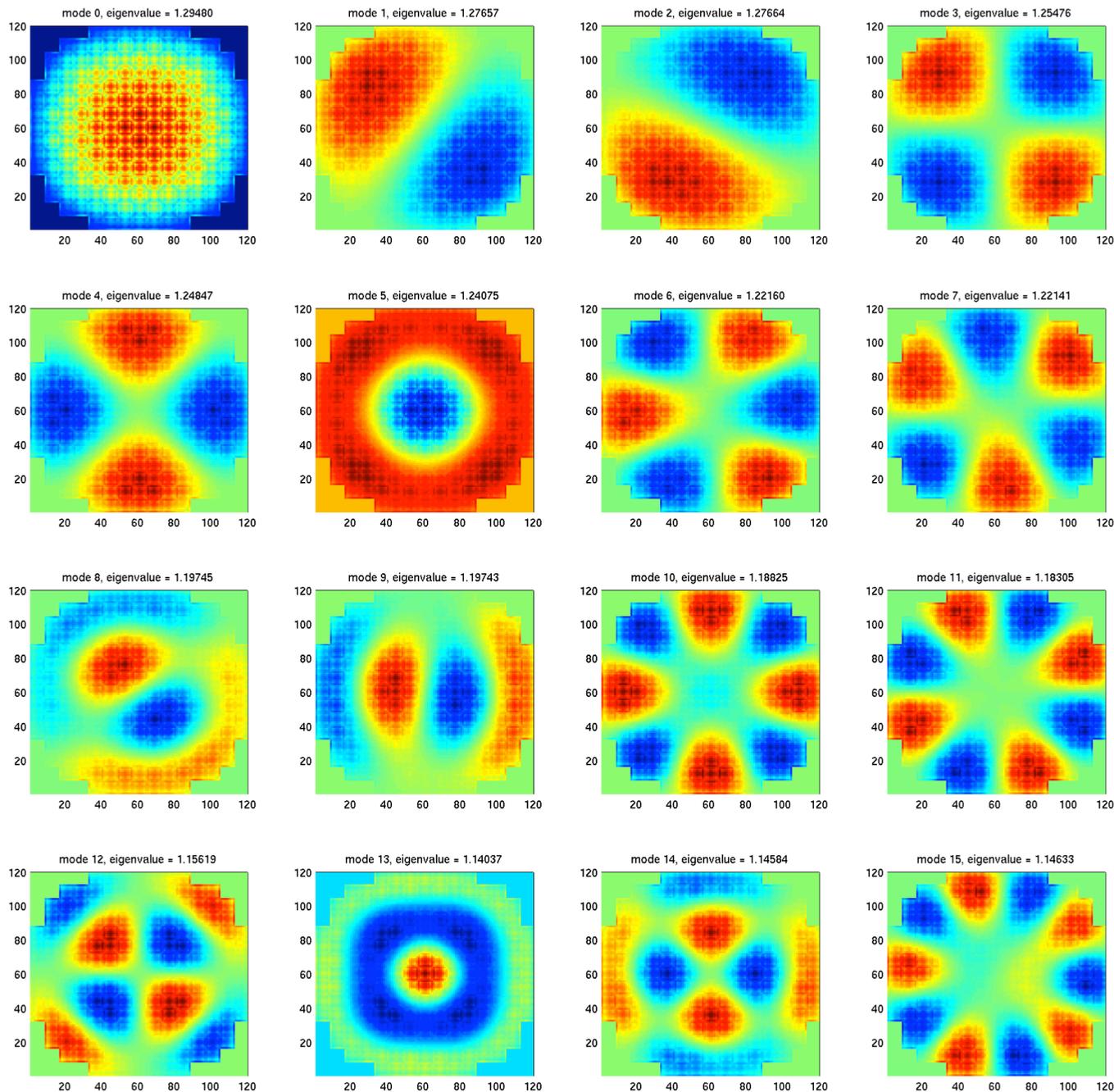
$$(k_p - k_q) \cdot (\vec{S}_p \cdot \vec{S}_q^\dagger) = 0$$

**F is nonsymmetric**

**$S_n$  is a right eigenvector of F,       $S_n^\dagger$  is a left eigenvector of F**

**$S_n$  and  $S_m^\dagger$  are biorthogonal**

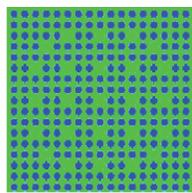
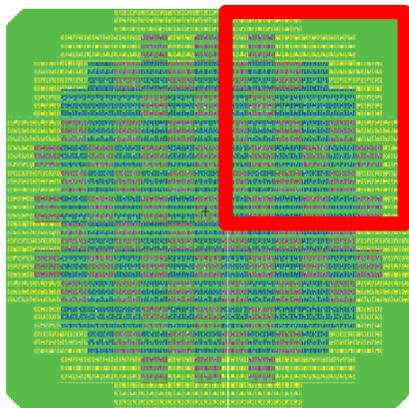
# PWR – Source Eigenmodes for 120x120x1 Spatial Mesh



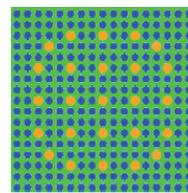
$n$	$K_n$
0	1.29480
1	1.27664
2	1.27657
3	1.25476
4	1.24847
5	1.24075
6	1.22160
7	1.22141
8	1.19745
9	1.19743
10	1.18825
11	1.18305
12	1.15619
13	1.14633
14	1.14617
15	1.14584

# PWR – with Perturbations

- Insert SS304 Control Rods in each assembly in quadrant of core



Original

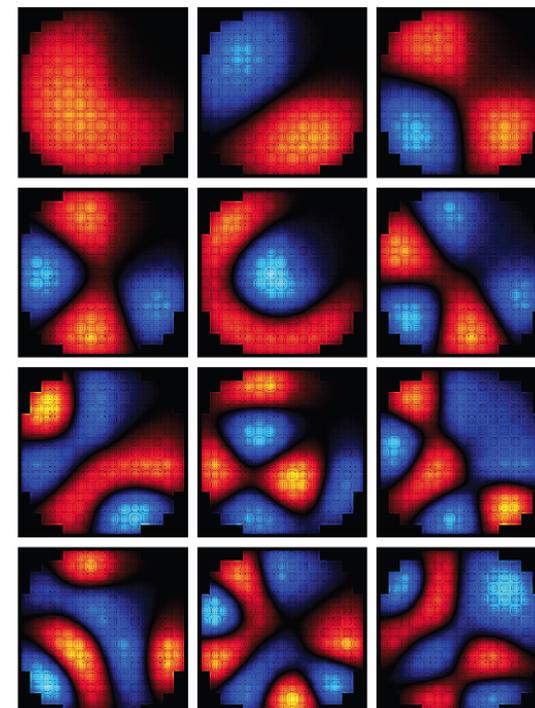
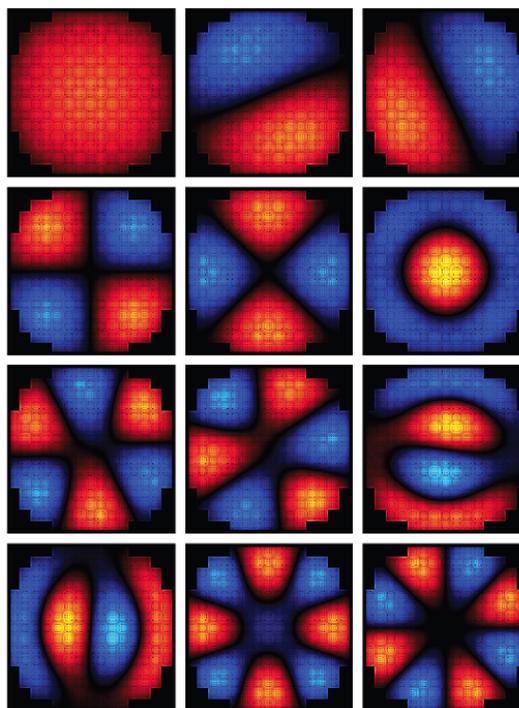


Perturbed

Original

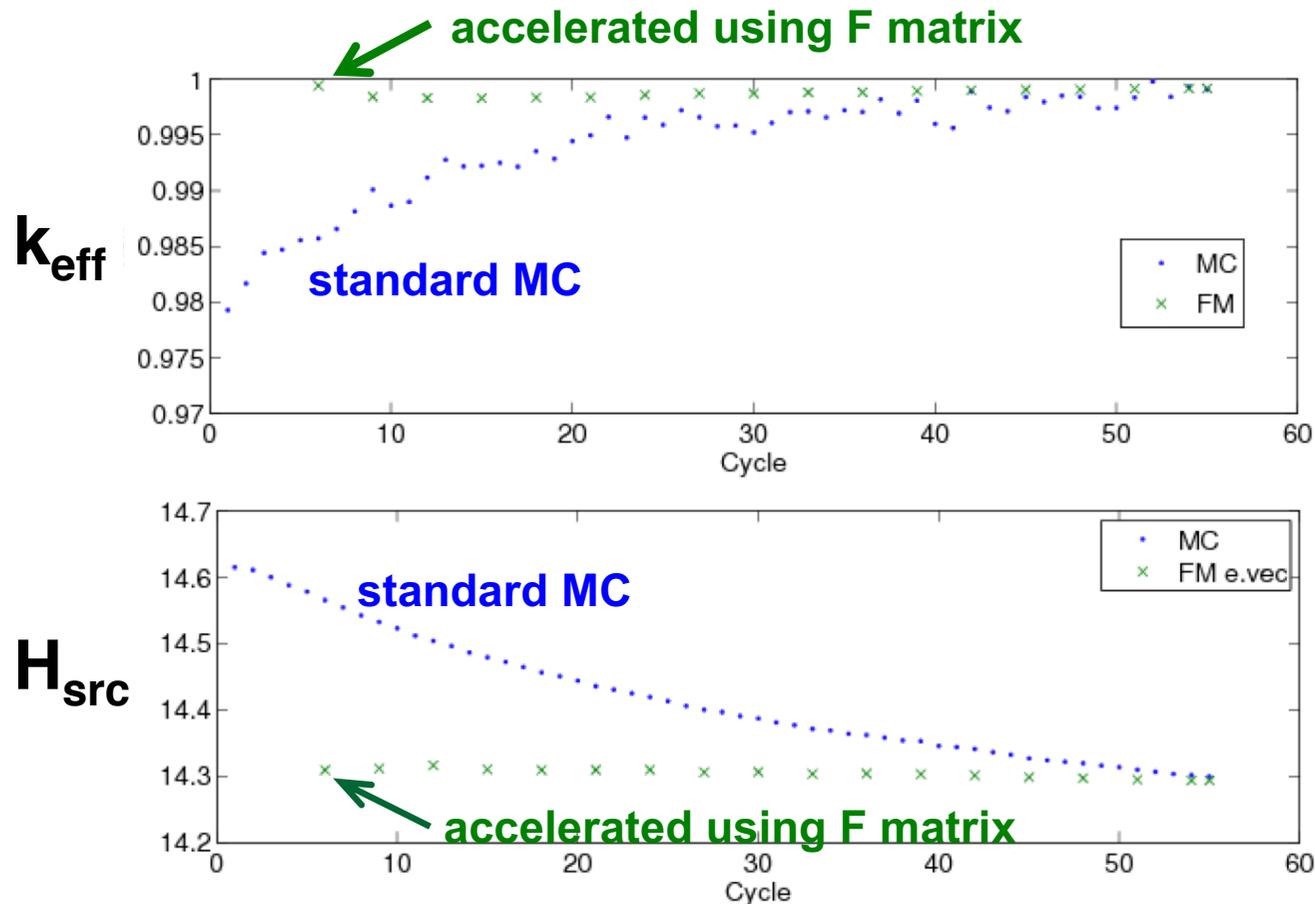
Perturbed

Fission Source  
Eigenmodes



## PWR - Convergence Acceleration Using Fission Matrix

- Fission matrix can be used to **accelerate convergence** of the MCNP neutron source distribution during inactive cycles
- Requires only fundamental forward mode
- Very impressive convergence improvement



# Fission Bank Reordering

# Reproducibility & Threading

---

- **For criticality problems with OpenMP threading, the fission-bank `fso()` needs to be reordered into a unique ordering that is independent of the number of threads or MPI processes.**
- **This was previously done by very crude, inefficient sorting, and did not scale well for large numbers of neutrons/cycle.**

**Scaling  $\sim O(N^2)$**

**$N = \text{neutrons/cycle}$**

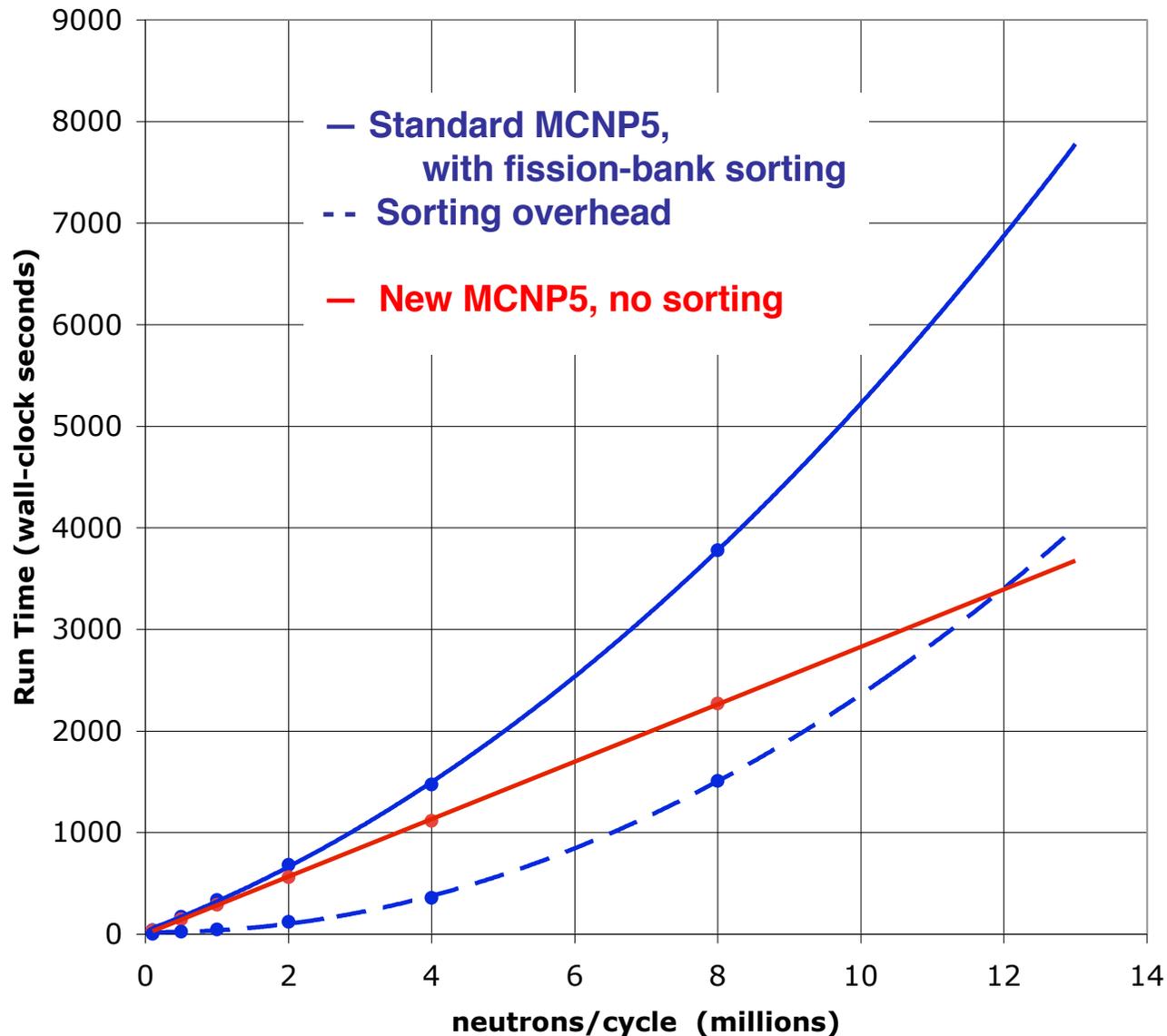
- **A new routine was added, `fso_reorder`, to provide a unique reordering of `fso()` **WITHOUT SORTING**. This is based on:**

FB Brown & TM Sutton, "Reproducibility and Monte Carlo Eigenvalue Calculations", Trans Am Nuc Soc 65, 235 (1992)

**Scaling  $\sim O(N)$**

**$N = \text{neutrons/cycle}$**

# Timing Studies



## PWR2D Model

- 1/4-core, detailed geometry, ENDF/B-VII
- Mac Pro, 3 GHz, 2 quad-core Xeon, 8 threads
- Identical results for old & new reordering
- **For >10M neutrs/cycle, old sorting took more time than running neutrs**
- **New scheme eliminates this, scales**

# Random Sampling Improvements

# Sampling the Evaporation Spectrum

- During the analysis of neutron collisions in MCNP, one of the possible Probability Density Functions (PDFs) for the outgoing neutron energy,  $E_{out}$ , is an “evaporation spectrum” (ENDF Law 9) given by:

$$f(E_{in} \rightarrow E_{out}) = C \cdot E_{out} \cdot e^{-E_{out}/T}, \quad 0 \leq E_{out} \leq E_{in} - U$$

Where  $T$  and  $U$  are tabulated functions of  $E_{in}$ , and  $C = T^{-2} \cdot [1 - e^{-(E_{in}-U)/T} \cdot (1 + \frac{E_{in}-U}{T})]^{-1}$

- The sampling scheme used in MCNP since the 1970s is:

$$E_{out} = -T \cdot \ln(\xi_1 \xi_2), \quad \text{reject \& repeat if } E_{out} > E_{in} - U$$

- This rejection scheme is extremely inefficient when  $(E_{in}-U)/T$  is small.
  - With some ENDF/B-VII data,  $(E_{in}-U)/T$  is sometimes smaller than 0.1, giving rise to rejection sampling efficiencies of 0.1 % or smaller
  - For some problems, mcnp6 may get stuck in this rejection sampling loop for minutes, hours, or even days.

# New Sampling Scheme

- It can readily be shown that a truncated Gamma PDF (the evaporation spectrum) can be efficiently sampled with a rejection scheme based on truncated exponentials:

Let  $x = E_{out} / T$ ,  $w = (E_{in} - U) / T$ ,  
and  $g = 1 - e^{-w}$ . Then,

$$(1) \tilde{E} = -\ln(1 - g\xi)$$

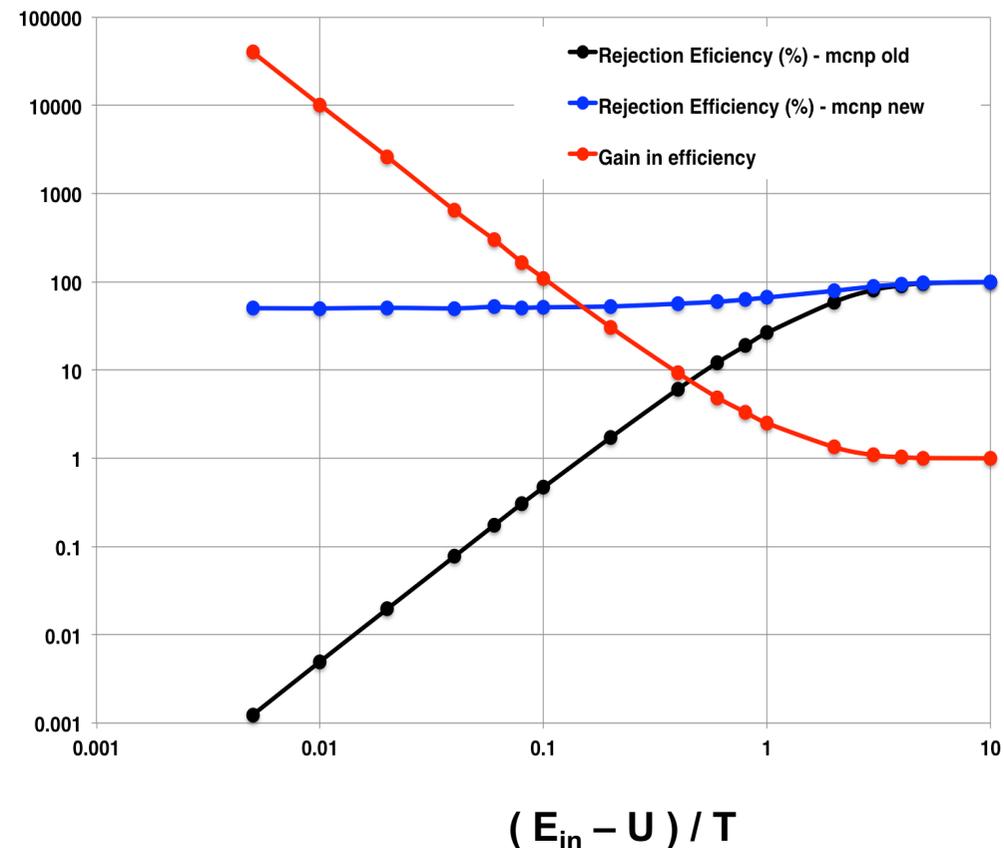
$$(2) \tilde{E}' = -\ln(1 - g\xi')$$

$$(3) x = \tilde{E} + \tilde{E}'$$

(3) Reject & resample if  $x > w$

$$(4) E_{out} = x \cdot T$$

- The efficiency of this rejection scheme is always greater than 50%, even for very small  $(E_{in} - U) / T$ . The gains in sampling efficiency can be very large, **1000x or more**:



# Summary & Conclusions

# Overall Performance Improvements

- **Initial 3-month effort, focus on speedup & optimization**
  - Focus on neutron criticality problems common to ASC & NCSP applications
  - **Speedups** from recent performance improvements

## Performance Test Set

### Criticality

ks1	1.76
ks2	2.13
ks3	1.35
ks4	1.36
baw1	2.19
baw2	1.59
fvf	2.04
g1	1.14
g2	2.20
pin	1.73

### Other

void1	3.03
void2	4.11
void2	4.11
void3	2.72
det1	1.67
med1	1.15
pht1	1.22

## Performance Benchmark Suite

### Speedups vs MCNP6.1 Release

### Neutron Problems

BAWXI2	4.37
GODIVA	1.05
Mode n in air w 750,000 tally bins	1.18
Well log problem	1.91
100M lattice cells in void	5.17

### Speedup

### Other

mode p e in air	1.01
mode n p e in air	1.05
mode p in air	1.20
Pulse height tally	1.20
Radiography	1.07

## VALIDATION\_CRITICALITY Suite

Measured wall-clock times, including data I/O:

mcnp5	release	34.7 min
mcnp6.1	release	43.9 min
mcnp6.1.1	NEW	27.9 min

→ **1.57 X** speedup over mcnp6.1

→ **1.24 X** speedup over mcnp5

# MC & Computing - Status

---

- **Computers continue to evolve - speed & accessibility**
  - Everyone now has multicore, Gflop computers - laptops, deskside
  - Almost everyone now has access to Linux clusters
  - New computers will have 16, 32, 48, 64, 80, ... cores per processor
- **MC codes must evolve**
  - **All MC codes - new & old - must be parallel, with threading + MPI**
  - Much larger problem sizes - millions of regions, materials, tallies
- **Monte Carlo for the 2020s & beyond:**
  - Outstanding success to date, will continue
  - More & more analysis will be done using Monte Carlo codes
  - New physic methods, eliminate approximations
  - Upgrade codes for huge problem sizes
  - New parallel computing algorithms
  - Improve robustness & ease-of-use

# Acknowledgments & References

---

## Acknowledgments

- Discussions of Monte Carlo algorithms and coding with George Zimmerman were, as always, stimulating and encouraging, and contributed significantly to the success of this work.
- This work was supported by the US DOE/NNSA Nuclear Criticality Safety Program and the Advanced Simulation & Computing Program.

## References

- F.B. Brown, B.C. Kiedrowski, J.S. Bull, "Verification of MCNP6.1 and MCNP6.1.1 for Criticality Safety Applications," Los Alamos National Laboratory report, LA-UR-14-22480 (2014).
- Forrest Brown, Brian Kiedrowski, Jeffrey Bull, "MCNP5-1.60 Release Notes", Los Alamos National Laboratory report, LA-UR-10-06235 (2010).
- J.T. Goorley, et al., "Initial MCNP6 Release Overview - MCNP6 version 1.0," Los Alamos National Laboratory report, LA-UR-13-22934 (2013).
- J.R. Tramm & A.R. Siegel, "Memory Bottlenecks and Memory Contention in Multi-Core Monte Carlo Transport Codes," Proceedings of SNA+MC 2013, Paris, France Oct 27-31 (2013).
- F.B. Brown, "Present Status of Vectorized Monte Carlo," *Trans. Am. Nucl. Soc.* 55, 323 (1987).
- J. Leppänen, "Two practical methods for unionized energy grid construction in continuous-energy Monte Carlo neutron transport calculation," *Ann. Nucl. Energy*, 36 (2009).
- J. Leppänen, A. Isotalo, "Burnup calculation methodology in the Serpent 2 Monte Carlo code," Proceedings of PHYSOR-2012, Knoxville, TN, Apr. 15-20 (2012).
- G. Zimmerman, private communication to F.B. Brown (2013).
- D. Austin, KAPL, private communication to F.B. Brown (~1989).





