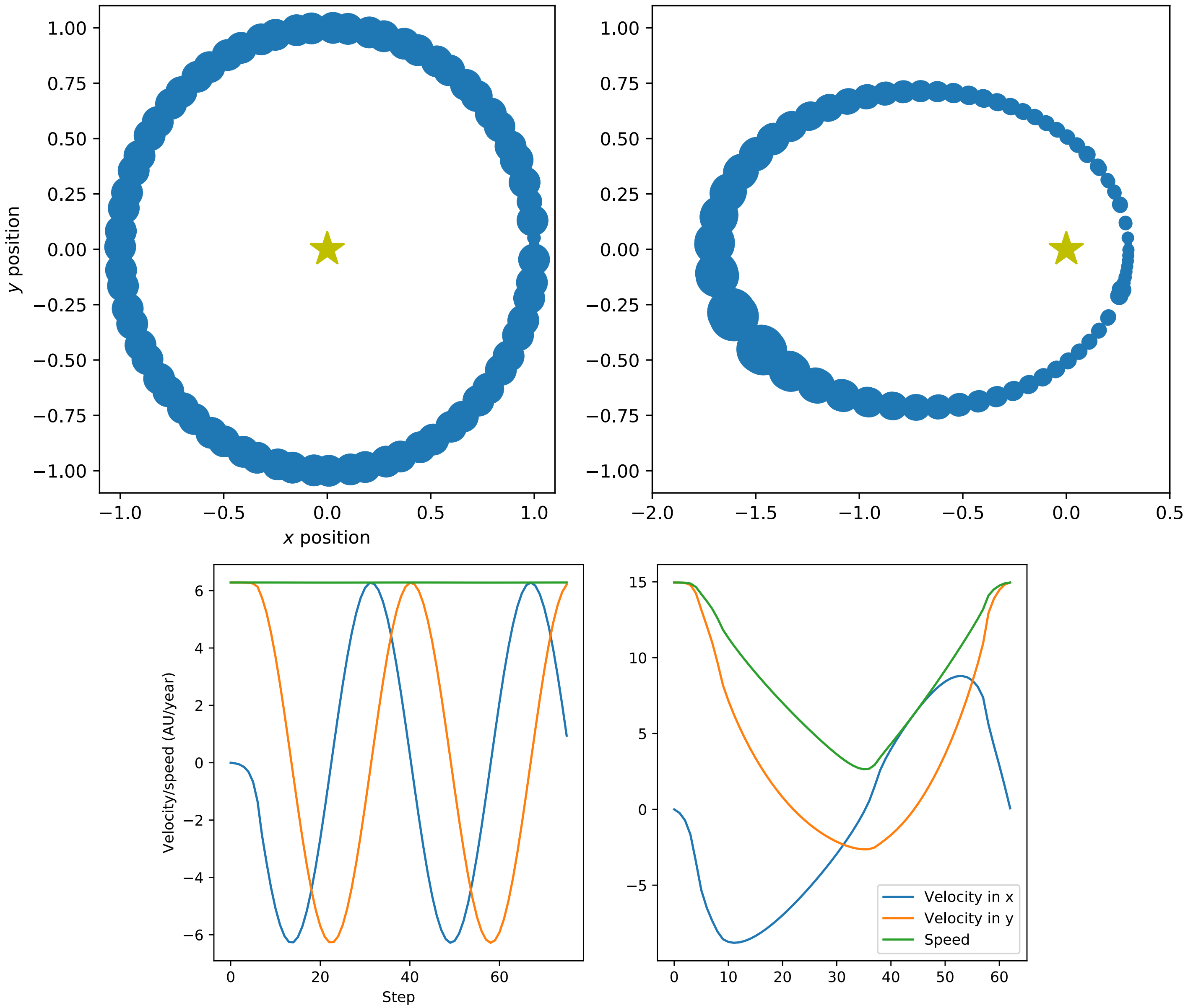
PHY604 Lecture 8

September 16, 2021

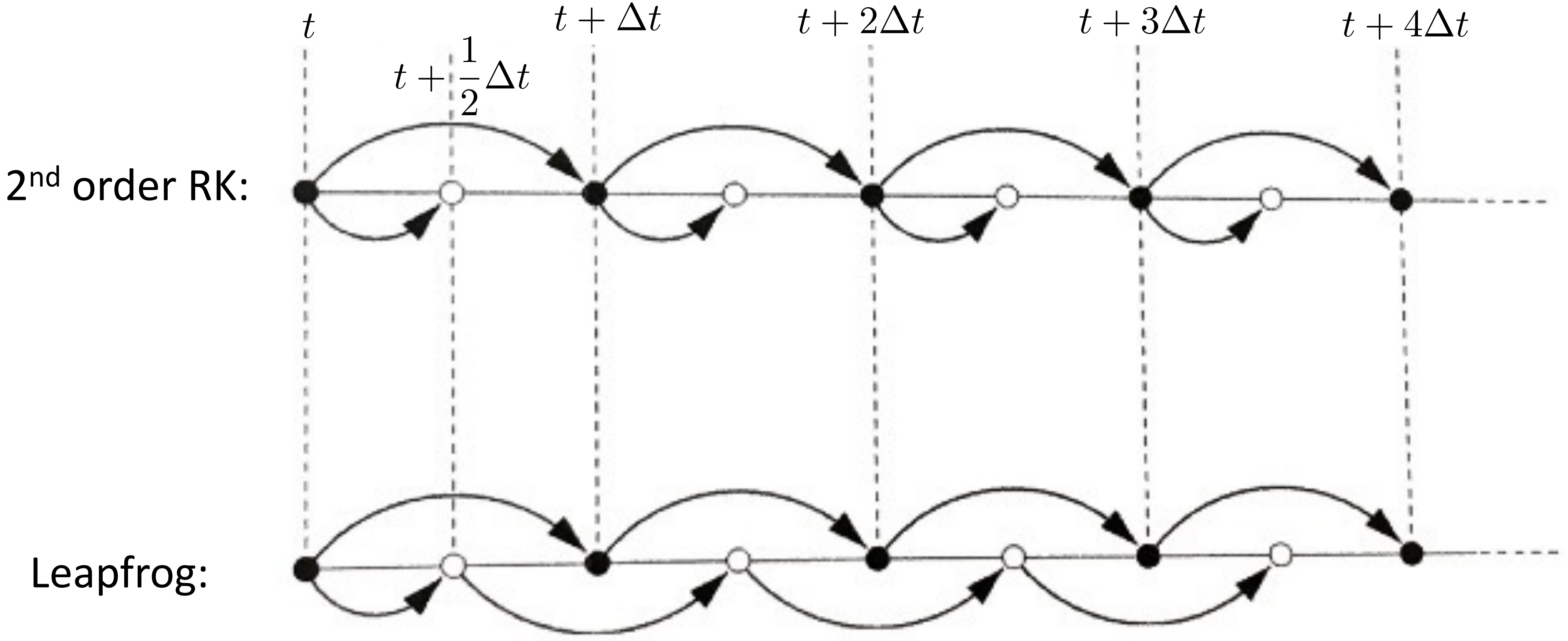
Review: Elliptical orbit with adaptive 4th-order RK

<u>Circular</u>:
x_0= 1 AU
v_{y0} =6.283185 AU/year



<u>Elliptical</u>:
x_0= 0.3 AU
v_{y0} =14.955378 AU/year

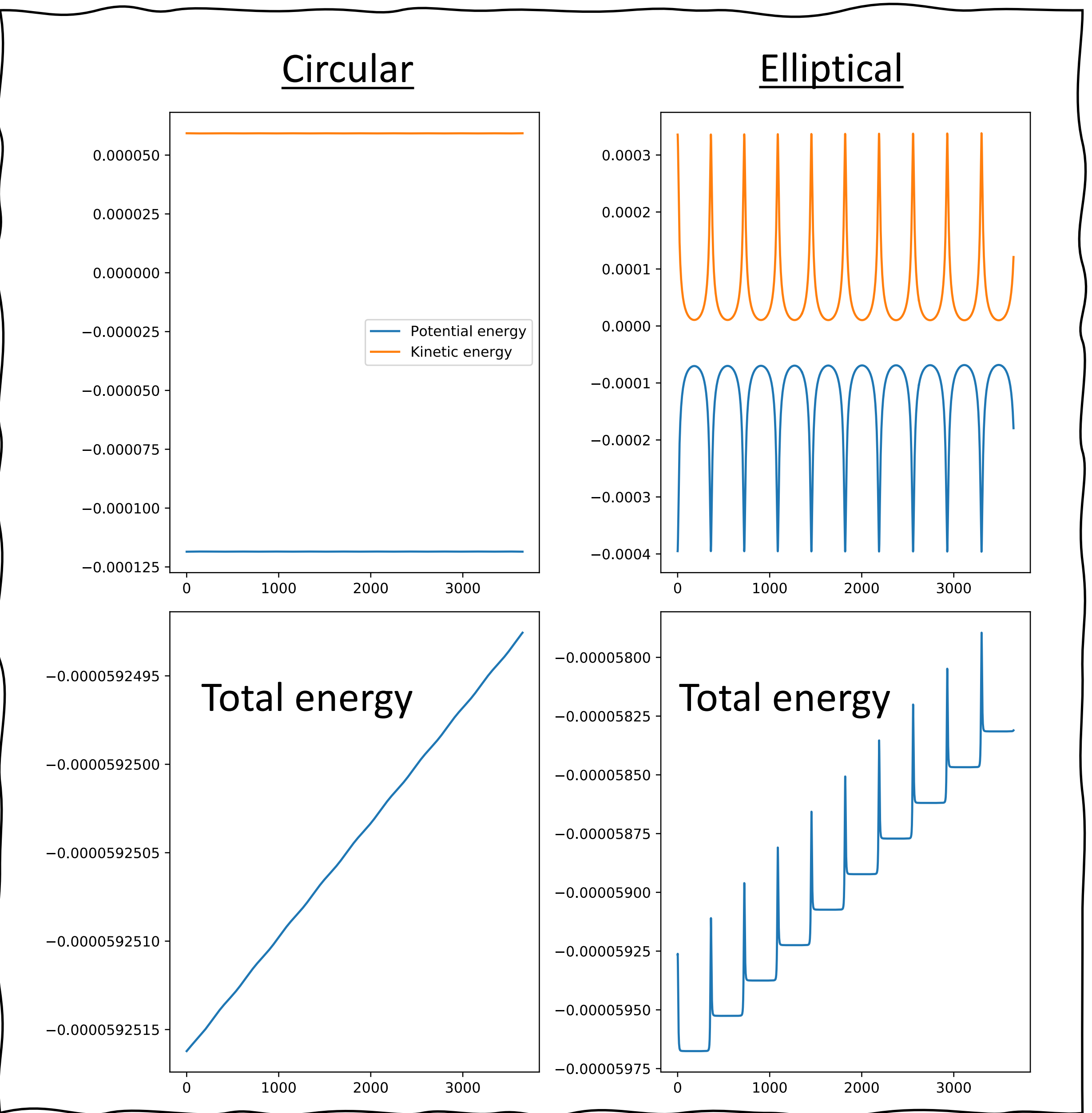
Review: Leapfrog method versus 2nd order RK



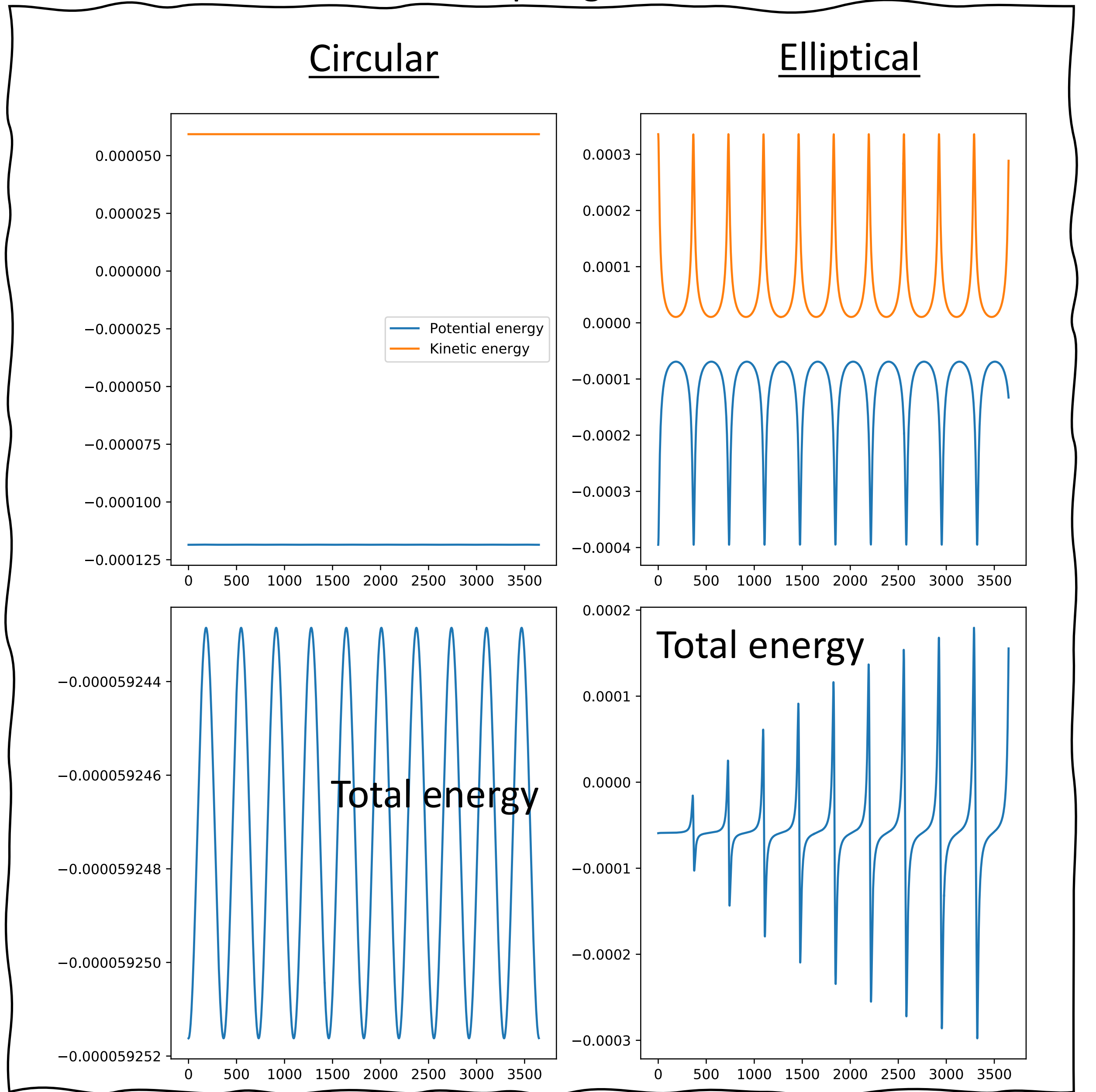
(Newman)

Review: Time-reversal symmetry and energy conservation!

2nd order RK



Leapfrog



Today's lecture:
ODEs and Linear Algebra

- Beyond RK: Other methods for ODEs
  - Bulirsch-Stoer Method
- Boundary Value problems
- Eigenvalue problems
- Begin discussing linear algebra

Bulirsch-Stoer Method

- Why do we care about the modified midpoint method and even-powered errors? They are the basis of the Bulirsch-Stoer Method

- This method combines the modified midpoint method with Richardson extrapolation (e.g., the Romberg method for integrals)

Simple example of Bulirsch-Stoer: First order ODE with one variable

- Equation: $\dfrac{dx}{dt} = f(x,t)$
- We would like to solve from t to t_f, with $x(t)$ given

- Start by using the modified midpoint method with a single step $\Delta t_1 = t_f$
  - More specifically, two half steps
  - Call this estimate $R_{1,1}$

- Now perform the calculation for $\Delta t_2 = 1/2\ t_f$ to get $R_{2,1}$

Performing Richardson extrapolation

- We can write the "exact" expressions since we know the form of the errors (using $\Delta t_1 = 2\Delta t_2$)

$$x(t+t_f) = R_{2,1} + c_1 \Delta t_2^2 + \mathcal{O}(\Delta t_2^4)$$

$$x(t+t_f) = R_{1,1} + c_1 \Delta t_1^2 + \mathcal{O}(\Delta t_1^4) = R_{1,1} + 4c_1 \Delta t_2^2 + \mathcal{O}(\Delta t_2^4)$$

- So:

$$c_1 \Delta t_2^2 = \frac{1}{3}(R_{2,1} - R_{1,1})$$

- And:

New estimate accurate to fourth order!

$$x(t+t_f) = \underbrace{R_{2,1} + \frac{1}{3}(R_{2,1} - R_{1,1})}_{R_{2,2}} + \mathcal{O}(\Delta t_2^4)$$

Performing Richardson extrapolation, cont.

- Let’s do another step: Calculate $R_{3,1}$ with Δt_3=1/3 t_f
- Following the same steps as before:

$$R_{3,2} = R_{3,1} + \frac{4}{5}(R_{3,1} - R_{2,1})$$

- Then we can write the “exact” result:

$$x(t+t_f) = R_{3,2} + c_2 \Delta t_3^4 + \mathcal{O}(\Delta t_3^6)$$

- From what we had previously:

$$x(t+t_f) = R_{2,2} + c_2 \Delta t_2^4 + \mathcal{O}(\Delta t_2^6) = R_{2,2} + \frac{81}{16} c_2 \Delta t_3^4 + \mathcal{O}(\Delta t_3^6)$$

- Equating these gives: $c_2 \Delta t_3^4 = \dfrac{16}{65}(R_{3,2} - R_{2,2})$

Performing Richardson extrapolation, cont.

- So, we have: $x(t+t_f) = \underbrace{R_{3,2} + \frac{16}{65}(R_{3,2} - R_{2,2})}_{R_{3,3}} + \mathcal{O}(\Delta t_3^6)$

New estimate accurate to sixth order!

- Where: $R_{3,3} = R_{3,2} + \frac{16}{65}(R_{3,2} - R_{2,2})$

- Three modified midpoint steps, and already have a sixth-order error
  - Gain two orders of accuracy with each step

General Richardson extrapolation

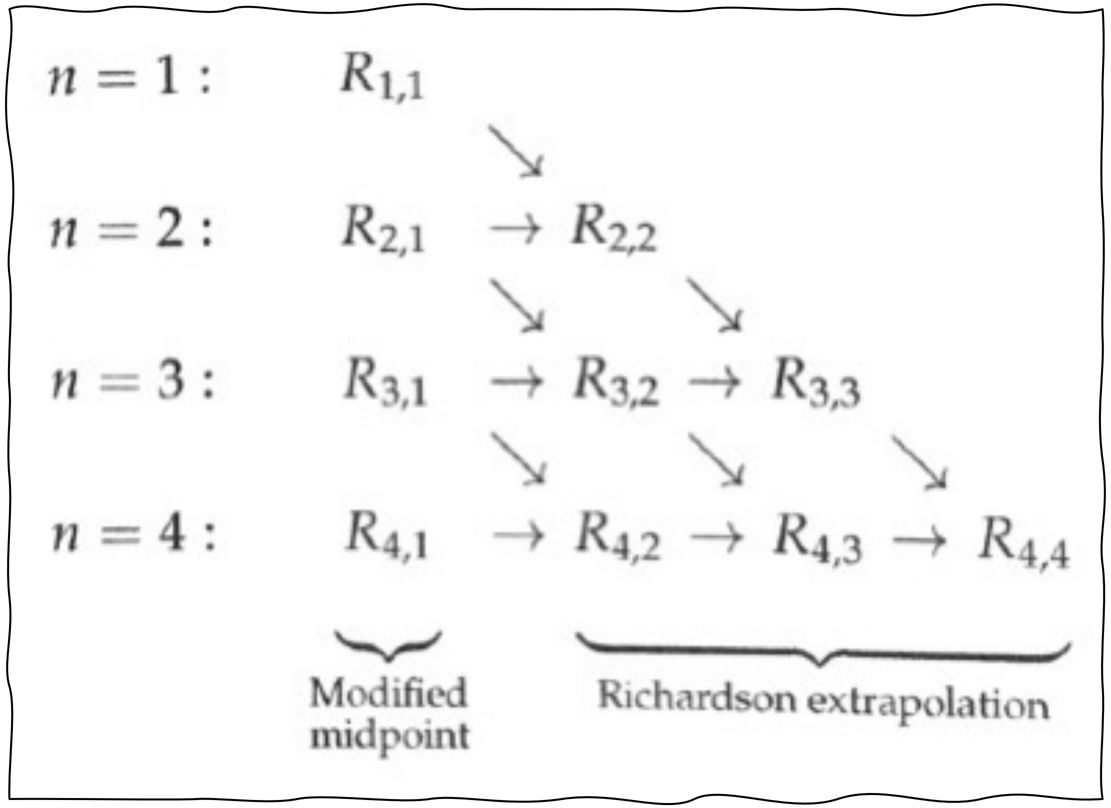
- n is the number of modified midpoint steps, which gives us $R_{n,1}$
  - Can obtain $R_{n,m}$ for $m < n$

$$R_{n,m+1} = R_{n,m} + \frac{R_{n,m} - R_{n-1,m}}{[n/(n-1)]^{2m} - 1}$$

  - See Newman Sec. 8.5

- Which gives an estimate of the result:

$$x(t + t_f) = R_{n,m+1} + \mathcal{O}(\Delta_n^{2m+2})$$



(Newman)

Comments about Bulirsch-Stoer

- Adaptive method: Provides error and estimate
  - Continue until error is below a given accuracy
- Similar approach to Romberg integration with some key differences
  - Increase number of intervals by one in BS instead of doubling in Romberg
  - Not possible to "reuse" previous points like in Romberg
- Only provides accurate estimate for final result $x(t+\mathrm{t}_f)$
  - At intermediate points, we just get raw midpoint method estimates (accurate to Δt^2)
  - Not well suited if we need many (100's or 1000's) steps, so only for rather small regions, where we can get accuracy with < 8 steps
- Can divide larger intervals into smaller ones and apply the BS method
- Can give better accuracy with less work then RK, especially for relatively smooth functions
  - RK should be used for ODEs with pathological behavior, large fluctuations, divergences, etc.

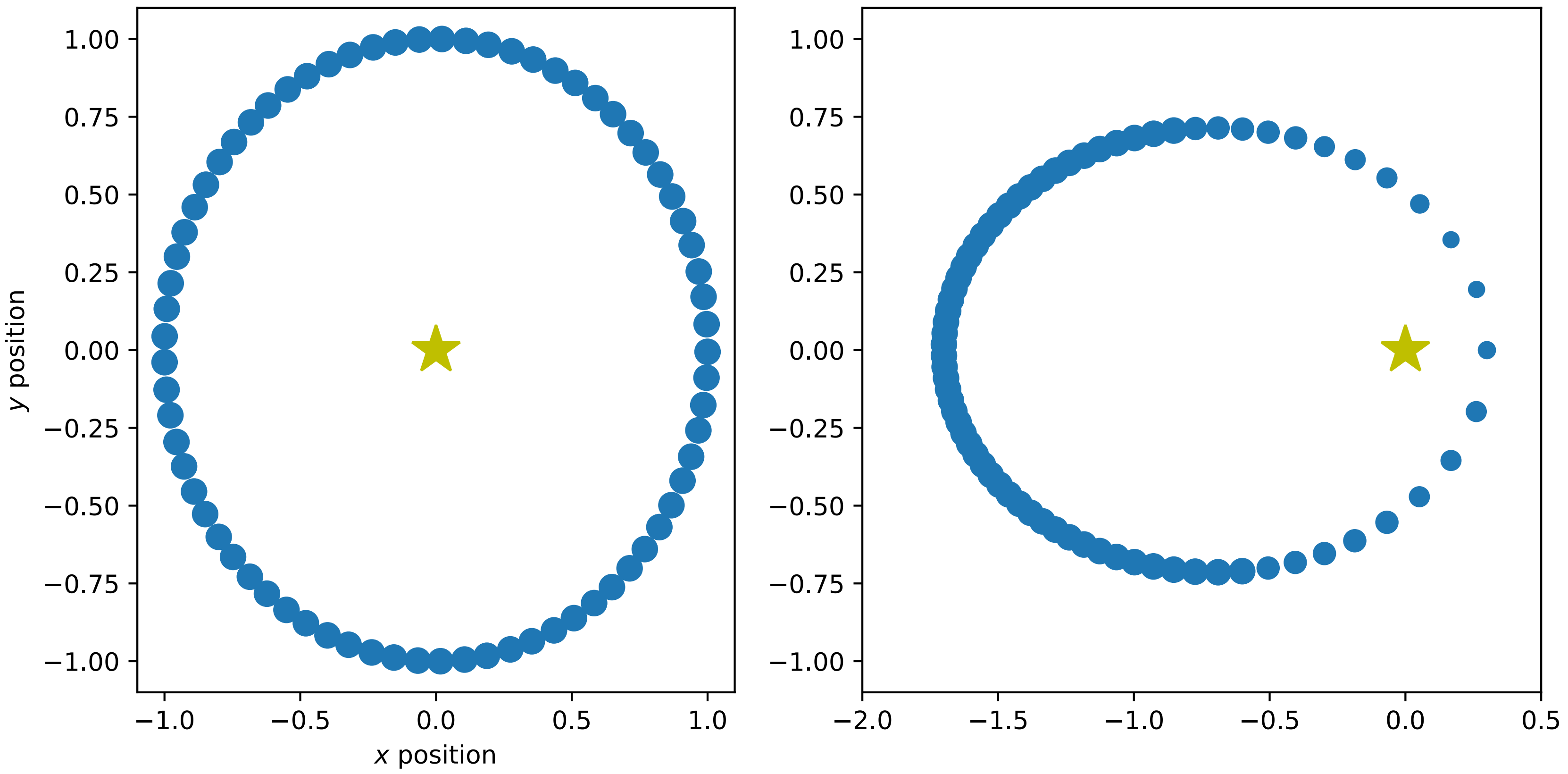
Bulirsch-Stoer Method: Summary

- Say we would like to solve an ODE from t to t_f up to accuracy δ per step
- First, divide the total range into N equal intervals of length t_H. Then do the following steps for each interval:

- 1. Perform a modified midpoint step with one interval from t to t_H to get $R_{1,1}$
- 2. Increase the number of intervals by one to n and calculate $R_{\mathrm{n},1}$ with the modified midpoint method
- 3. Calculate the "row" via Richardson extrapolation, i.e., $R_{n,2} \ldots R_{n,n}$
- 4. Compare the error to the target accuracy δt_H. If it is larger than the target accuracy, return to step 2. If it is less than the target accuracy, go to the next interval.

D

Example: Orbits with the Bulirsch-Stoer method



Today's lecture:
ODEs and Linear Algebra

- Beyond RK: Other methods for ODEs
  - Bulirsch-Stoer Method

- Boundary Value problems

- Eigenvalue problems

- Begin discussing linear algebra

Boundary value problems

- The orbital example we have been studying is an initial value problem: Solving ODEs given some initial value
- Boundary value problems: Conditions needed to specify the solution given at some different (or additional) points to the initial point
  - E.g.: Find a solution for the EOM such that the trajectory passes through a specific point in the future
- Boundary value problems are more difficult to solve
  - Two methods: Shooting method and relaxation method (we will discuss the latter in terms of PDEs later)

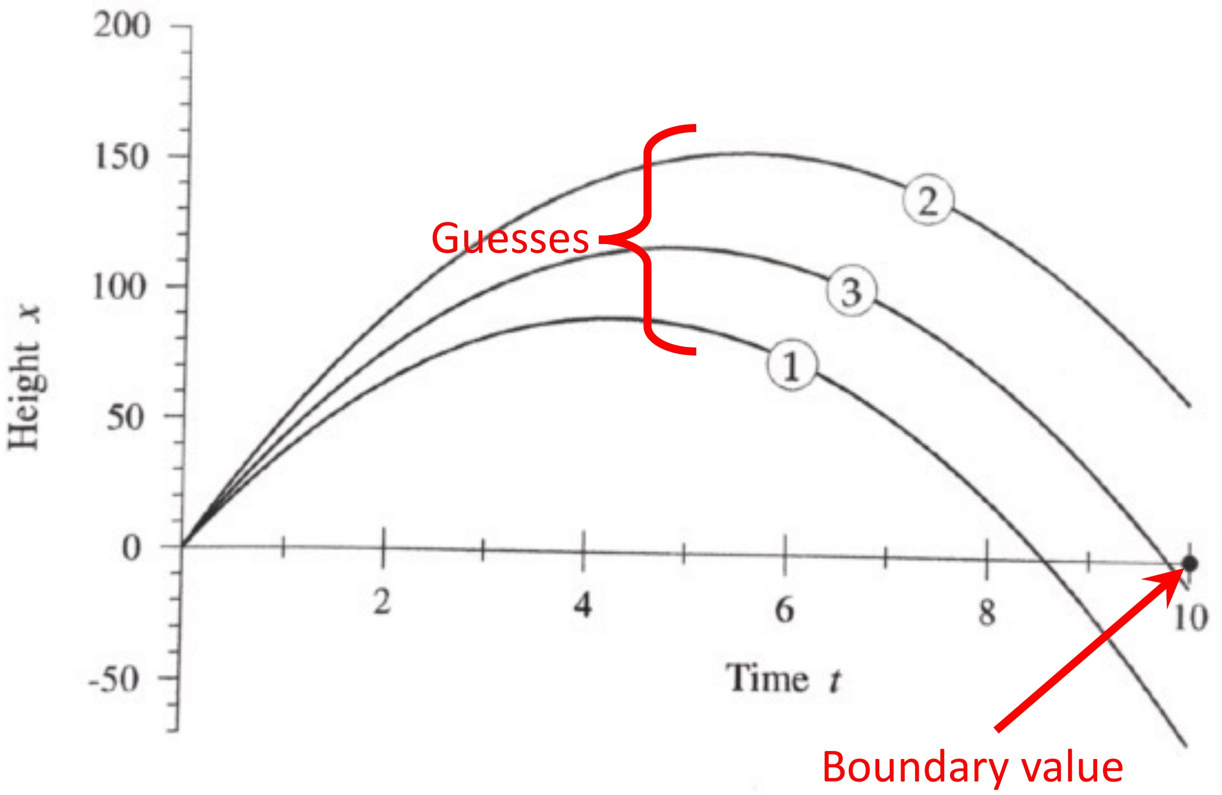
Shooting method example: Ball thrown in the air

- "Trial-and-error" method: Searches for correct values of initial conditions that match a given set of boundary conditions

- Example (from Newman Sec. 8.6): Height of a ball thrown in the air

$$\frac{d^2x}{dt^2} = -G$$

- Guess initial conditions (initial vertical velocity) for which the ball will return to the ground at a given time *t*



How do we modify initial conditions between guesses?

- Write the height of the ball at the boundary t_1 as $x = f(v)$ where v is the initial velocity
- If we want the ball to be at $x = 0$ at t_1, we need to solve $f(v) = 0$

- So, we have reformulated the problem as finding a root of a function
  - We can use, e.g., the bisection method, Newton-Raphson method, secant method
- The function is "evaluated" by solving the differential equation
  - We can use any method discussed previously, e.g., Runge-Kutta, Bulirsch-Stoer, etc.

Today’s lecture:
ODEs and Linear Algebra

- Beyond RK: Other methods for ODEs
  - Bulirsch-Stoer Method
- Boundary Value problems
- Eigenvalue problems
- Begin discussing linear algebra

Eigenvalue problems

- Special type of boundary value problem: Linear and homogeneous
  - Every term is linear in the dependent variable
- E.g.: Schrodinger equation:

$$-\frac{\hbar}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x)$$

- Consider the Schrodinger equation in a 1D square well with infinite walls:

$$V(x) = \begin{cases} 0, & \text{for } 0 < x < L \\ \infty, & \text{elsewhere} \end{cases}$$

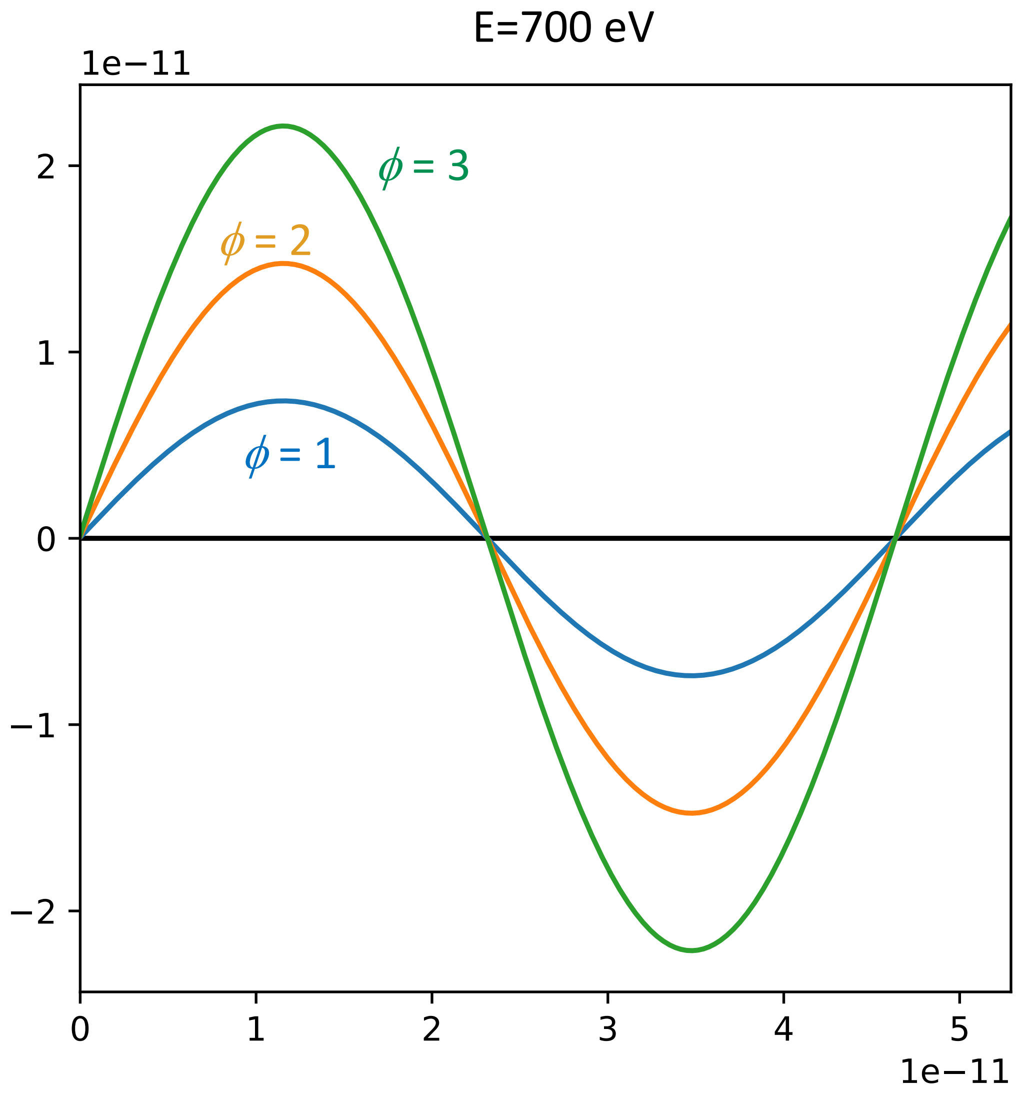
Schrodinger equation in 1D well

- As usual, make into system of 1D ODEs:

$$\frac{d\psi}{dx} = \phi, \qquad \frac{d\phi}{dx} = \frac{2m}{\hbar^2}[V(x) - E]\psi$$

- Know that $\psi = 0$ at $x = 0$ and $x = L$, but don't know ϕ
- Let's choose a value of E and solve using some choices for ϕ :
- Since the equation is linear, scaling the initial conditions exactly scales the $\psi(x)$

- **No matter what ϕ, we will never get a valid solution!** (only affects overall magnitude, not shape)



Only specific E has a valid solution

- Solutions only exist for eigenvalues
- Need to vary E, ϕ can be fixed via normalization
- Same strategy, Find the E such that $\psi(L) = 0$

Today's lecture:
ODEs and Linear Algebra

- Beyond RK: Other methods for ODEs
  - Bulirsch-Stoer Method

- Boundary Value problems

- Eigenvalue problems

- Begin discussing linear algebra

Numerical linear algebra (Garcia Ch. 4)

- Basic problem to solve: **A x** = **b**

- We have already seen many cases where we need to solve linear systems of equations
  - E.g., ODE integration, cubic spline interpolation

- More that we will come across:
  - Solving the diffusion PDE
  - Multivariable root-finding
  - Curve fitting

- We will explore some key methods to understand what they do
  - Mostly, efficient and robust libraries exist, so no need to reprogram

- Often it is illustrative to compare between how we would solve linear algebra by hand and (efficiently) on the computer

Review of matrices: Multiplication

- Matrix-vector multiplication:
  - **A** is *m* x *n* matrix
  - **x** is *n* x 1 (column) vector
  - Result: **b** is *m* x 1 (column vector)
  - Simple scaling: $O(N^2)$ operations

$$b_i = (Ax)_i = \sum_{j=1}^{n} A_{ij} x_j$$

- Matrix-matrix multiplication
  - **A** is *m* x *n* matrix
  - **B** is *n* x *p* matrix
  - Result: **AB** is *m* x *p* matrix
  - Direct multiplication: $O(N^3)$ operations
    - Some faster algorithms exist (make use of organization of sub-matrices for simplification)

$$(AB)_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Review of matrices: Determinant

- Encodes some information about a square matrix
  - Used in come linear systems algorithms
  - Solution to linear systems only exists if determinant is nonzero
- Simple algorithm for obtaining determinant is Laplace expansion
- For simple matrices, can be done by hand:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

- What about big matrices?

Review of matrices: Determinant

- Encodes some information about a square matrix
  - Used in come linear systems algorithms
  - Solution to linear systems only exists if determinant is nonzero
- **By hand:** Simple algorithm for obtaining determinant is Laplace expansion
- For simple matrices, can be done by hand:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

- What about big matrices? **Will need a more efficient implementation!**

Review of matrices: Inverse

- $\mathbf{A}^{-1}\mathbf{A}=\mathbf{A}\mathbf{A}^{-1}=\mathbf{I}$

- Formally, the solution to a linear system $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ is $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$
  - Usually less expensive to get the solution without computing the inverse first

- Non-invertible (i.e., singular) if determinant is 0

By hand: Cramer’s rule

- One simple way to solve **A x** = **b** is:

$$x_i = \frac{|\mathbf{A}_i|}{|\mathbf{A}|}$$

- Where $\mathbf{A}_i$ is **A** with the *i*th column replaced by **b**

- Comparable speed to calculating the inverse

By hand: Gaussian elimination

- Main general technique for solving **A x = b**
  - Does not involve matrix inversion
  - For “special” matrices, faster techniques may apply
- Involves forward-elimination and back-substitution

- Consider a simple example (from Garcia Ch. 4):

$$
\begin{aligned}
x_1 + x_2 \quad + x_3 &= 6 \\
-x_1 + 2x_2 \qquad &= 3 \\
2x_1 \qquad\quad + x_3 &= 5
\end{aligned}
$$

By hand: Forward elimination

- 1. Eliminate x_1 from second and third equation. Add first equation to the second and subtract twice the first equation from the third:

$$\begin{aligned} x_1+x_2 \quad +x_3 &= 6 \\ 3x_2+x_3 &= 9 \\ -2x_2-x_3 &= -7 \end{aligned}$$

- 2. Eliminate x_2 from third equation. Multiply the second equation by (-2/3) and subtract it from the third

$$\begin{aligned} x_1+x_2 \quad +x_3 &= 6 \\ 3x_2 \quad +x_3 &= 9 \\ -\frac{1}{3}x_3 &= -1 \end{aligned}$$

By hand: Back substitution

$$\begin{aligned} x_1 + x_2 \quad + x_3 &= 6 \\ 3x_2 \; + x_3 &= 9 \\ -\frac{1}{3}x_3 &= -1 \end{aligned}$$

- 3. Solve for $x_3 = 3$.
- 4. Substitute x_3 into the second equation to get $x_2 = 2$
- 5. Substitute x_3 and x_2 into the first equation to get $x_1 = 1$

- In general, for N variables and N equations:
  - Use forward elimination make the last equation provide the solution for x_N
  - Back substitute from the Nth equation to the first
  - Scales like N^3 (can do better for "sparse" equations)

Pitfalls of Gaussian substitution: Roundoff errors

- Consider a different example (also from Garcia):

$$\begin{aligned} \epsilon x_1 + x_2 + x_3 &= 5 \\ x_1 + x_2 \quad\quad &= 3 \\ x_1 + \quad + x_3 &= 4 \end{aligned}$$

- First, lets take $\epsilon \to 0$ and solve:

<u>Subtract second from third</u>:

$$\begin{aligned} x_2 + x_3 &= 5 \\ x_1 + x_2 \quad\quad &= 3 \\ -x_2 + x_3 &= 1 \end{aligned}$$

<u>Add first to third</u>:

$$\begin{aligned} x_2 + x_3 &= 5 \\ x_1 + x_2 \quad\quad &= 3 \\ 2x_3 &= 6 \end{aligned}$$

<u>Back substitute</u>:

$$\begin{aligned} x_2 &= 2 \\ x_1 &= 1 \\ x_3 &= 3 \end{aligned}$$

Roundoff error example: Now solve with ε

- Forward elimination starts by multiplying first equation by $1/\varepsilon$ and subtracting it from second and third:

$$\begin{aligned} \epsilon x_1 + \quad & x_2 \quad & +x_3 &= 5 \\ & (1 - 1/\epsilon) x_2 \quad & -(1/\epsilon) x_3 &= 3 - 5/\epsilon \\ & -(1/\epsilon) x_2 \quad & +(1 - 1/\epsilon) x_3 &= 4 - 5/\epsilon \end{aligned}$$

- Clearly have an issue if ε is near zero, e.g., if $C - 1/\epsilon \to -1/\epsilon$ for C order unity:

$$\begin{aligned} \epsilon x_1 + \quad & x_2 \quad & +x_3 &= 5 \\ & -(1/\epsilon) x_2 \quad & -(1/\epsilon) x_3 &= -5/\epsilon \\ & -(1/\epsilon) x_2 \quad & -(1/\epsilon) x_3 &= -5/\epsilon \end{aligned}$$

Cannot solve, now have two equations, three unknowns

Simple fix: Pivoting

- Interchange the order of the equations before performing the forward elimination

$$\begin{aligned} x_1 + x_2 &= 3 \\ \epsilon x_1 + x_2 + x_3 &= 5 \\ x_1 + + x_3 &= 4 \end{aligned}$$

- Now the first step of forward elimination gives us:

$$\begin{aligned} x_1 + x_2 &= 3 \\ (1-\epsilon)x_1 \quad + x_3 &= 5 - 3\epsilon \\ -x_2 + x_3 &= 1 \end{aligned}$$

- Now we round off:

$$\left.\begin{aligned} x_1 + x_2 &= 3 \\ x_1 \quad + x_3 &= 5 \\ -x_2 + x_3 &= 1 \end{aligned}\right\}$$

Same as when we initially took ε to 0.

Gaussian elimination with pivoting

- Partial-pivoting:
  - Interchange of rows to move the one with the largest element in the current column to the top
  - (Full pivoting would allow for row and column swaps—more complicated)
- Scaled pivoting
  - Consider largest element relative to all entries in its row
  - Further reduces roundoff when elements vary in magnitude greatly
- Row echelon form: This is the form that the matrix is in after forward elimination

Matrix determinants with Gaussian elimination

- Once we have done forward substitution and obtained a row echelon matrix it is trivial to calculate the determinant:

$$\det(\mathbf{A}) = (-1)^{N_{\mathrm{pivot}}} \prod_{i=1}^{N} A_{ii}^{\mathrm{row\text{-}echelon}}$$

- Every time we pivoted in the forward substitution, we change the sign

Matrix inverse with Gaussian elimination

- We can also use Gaussian elimination to fin the inverse of a matrix
- We would like to find **AA**$^{-1}$ = **I**
- We can use Gaussian elimination to solve: **A x**$_i$ = **e**$_i$
  - **e**$_i$ is a column of the identity:

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}, \quad \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix}, \ldots, \; \mathbf{e}_N = \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

  - **x**$_i$ is a column of the inverse:

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \ldots & \mathbf{x}_N \end{bmatrix}$$

Singular matrix

- If a matrix has a vanishing determinant, then the system is not solvable
- Common way for this to enter, one equation in the system is a linear combination of some others
- Not always easy to detect from the start

Singular and close to singular matrices

- Condition number: Measures how close to singular we are
  - How much **x** would change with a small change in **b**

$$\mathrm{cond}(\mathbf{A}) = ||\mathbf{A}||\ ||\mathbf{A}^{-1}||$$

  - Requires defining a norm of **A**
    - https://en.wikipedia.org/wiki/Matrix_norm
  - See, e.g., numpy implementation:
    - https://numpy.org/doc/stable/reference/generated/numpy.linalg.cond.html

- Rule of thumb: $\dfrac{||\mathbf{x}^{\mathrm{exact}} - \mathbf{x}^{\mathrm{calc}}||}{||\mathbf{x}^{\mathrm{exact}}||} \simeq \mathrm{cond}(\mathbf{A}) \cdot \epsilon^{\mathrm{machine}}$

After class tasks

- Homework 1 due Today by 11pm
  - Note the email about problem 4, your program just needs to work for $a > 1$
  - Office hours today 11:05am to 1:00pm

- Readings:
  - Newman Ch. 8
  - Garcia Ch. 4