PHY407: Computational Physics Fall, 2017

Instructor: Paul Kushner

TAs: Heather Fong, Alex Cabaj, Haruki Hirasawa

Lecture 3

- Integration by Gaussian Quadrature
- Numerical differentiation
- Solving linear systems

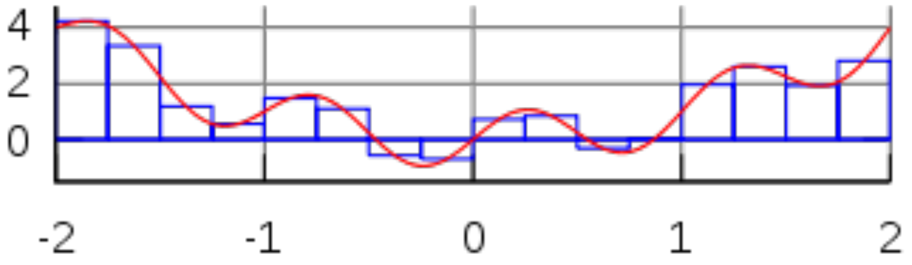
What are some sources of error in computational physics?

- Input errors
- Errors in setting up the discretized mathematical model (conceptual/mathematical)
- Algorithm errors
  - Roundoff/numerical
  - $\boxed{\text{– Approximation errors}}$ This week.
  - Ill-conditioned methods (instability)
- Output errors

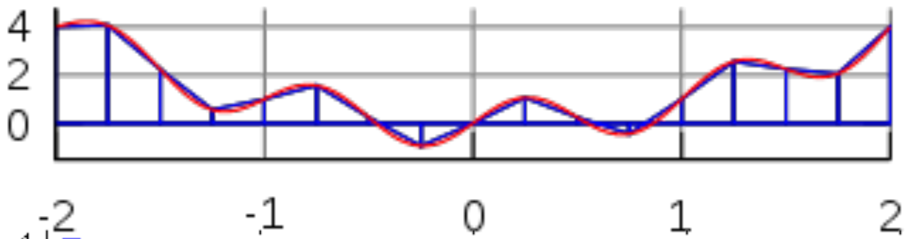
Newton-Cotes formulas

- Trapezoid and Simpson’s Rules are similar and can be generalized.
- Break your interval into small equal sub-intervals
- Approximate your function by a polynomial of some degree on that sub-interval.
- This class of methods leads to Newton-Cotes formulas

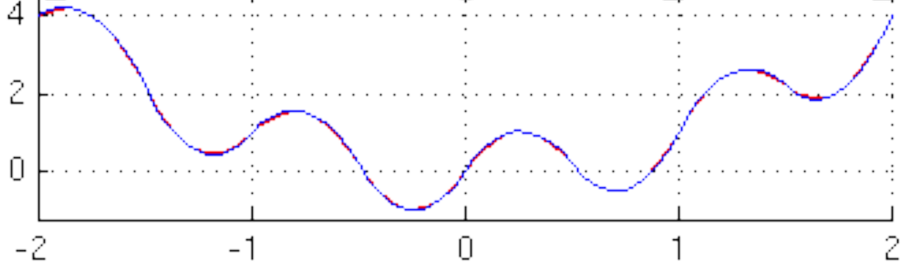
Mid-point rule: degree 0:



Trapezoidal rule: degree 1:



Simpson’s rule: degree 2:



…

N-C formulas

- All Newton-Coates formulas can be written in the form:

$$\boxed{\int_a^b f(x)dx \approx \sum_{k=1}^{N} \mathrm{w}_k f(x_k)}$$

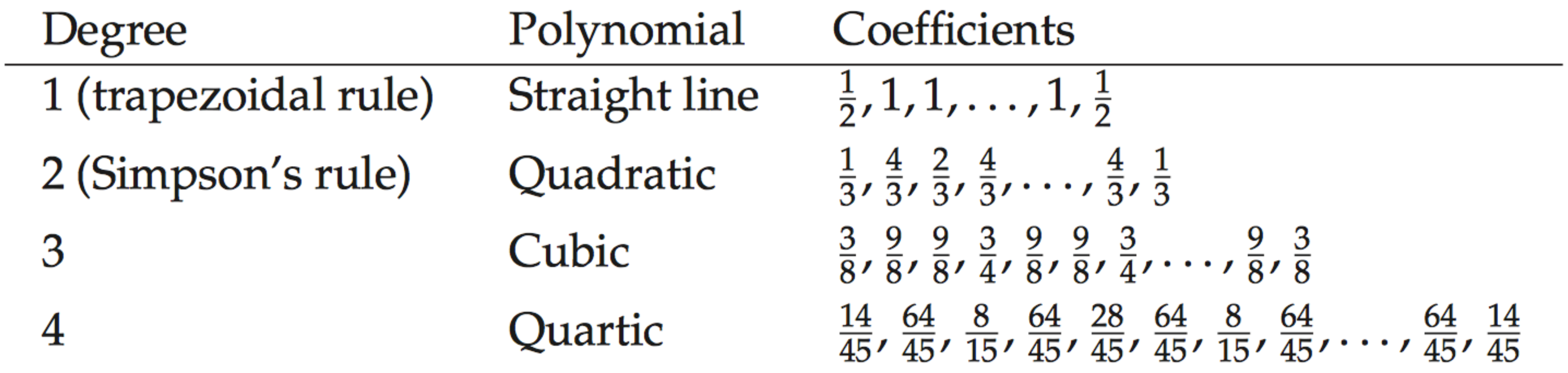
- w_k are called the “weights”. x_k are called the “sample points”. Notice above we are using “N” points to sample.

- N-C formulas of degree “N” are **exact for polynomials of degree N (which require N+1 points to determine)** (e.g. Trapezoid rule exact for linear functions, Simpson’s rule exact for quadratic functions).

- For N-C formulas, the sample points are **evenly spaced**.

Newton-Cotes formulas

- The table below shows the generalization of the rules we have seen, so far, to higher-order polynomials.



Degree	Polynomial	Coefficients
1 (trapezoidal rule)	Straight line	$\frac{1}{2}, 1, 1, \ldots, 1, \frac{1}{2}$
2 (Simpson's rule)	Quadratic	$\frac{1}{3}, \frac{4}{3}, \frac{2}{3}, \frac{4}{3}, \ldots, \frac{4}{3}, \frac{1}{3}$
3	Cubic	$\frac{3}{8}, \frac{9}{8}, \frac{9}{8}, \frac{3}{4}, \frac{9}{8}, \frac{9}{8}, \frac{3}{4}, \ldots, \frac{9}{8}, \frac{3}{8}$
4	Quartic	$\frac{14}{45}, \frac{64}{45}, \frac{8}{15}, \frac{64}{45}, \frac{28}{45}, \frac{64}{45}, \frac{8}{15}, \frac{64}{45}, \ldots, \frac{64}{45}, \frac{14}{45}$

Today: Gaussian Quadrature

- Allow your interval to be broken up into small *un-equal* sub-intervals

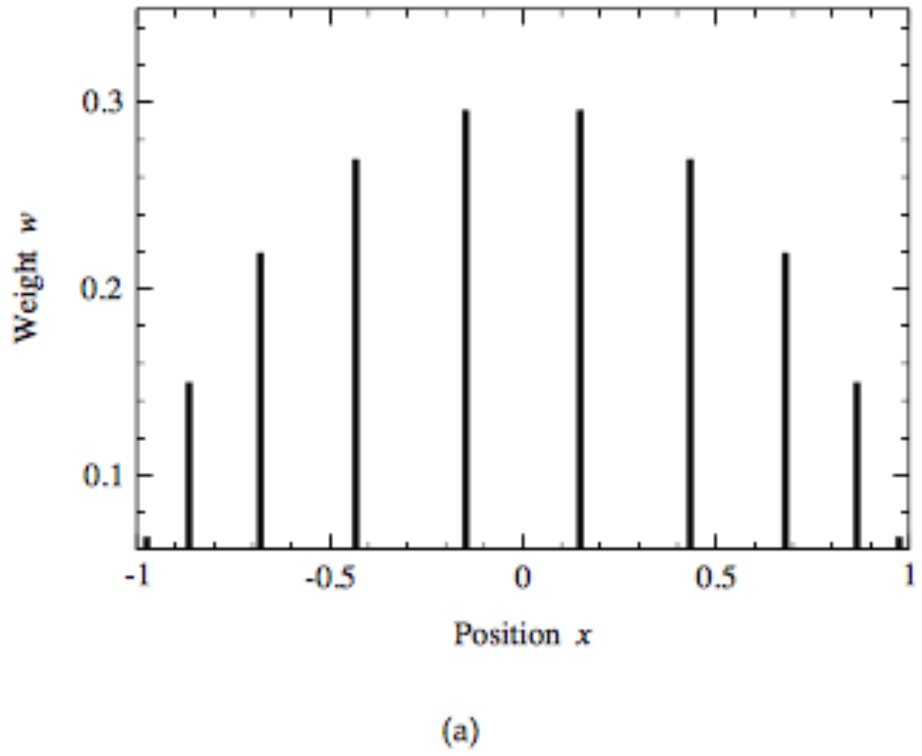
- Formula still has the form:

$$\boxed{\int_a^b f(x)dx \approx \sum_{k=1}^{N} \mathrm{w}_k f(x_k)}$$

- But now the sample points are **not evenly spaced**

- Because you have the freedom to set the locations of the x_k's and the w_k's, you can create an integration rule that **is exact for polynomials of up to order 2N-1** (rather than just N-1 for N-C formulas).
- So these can be MUCH more accurate.

Today: Gaussian Quadrature



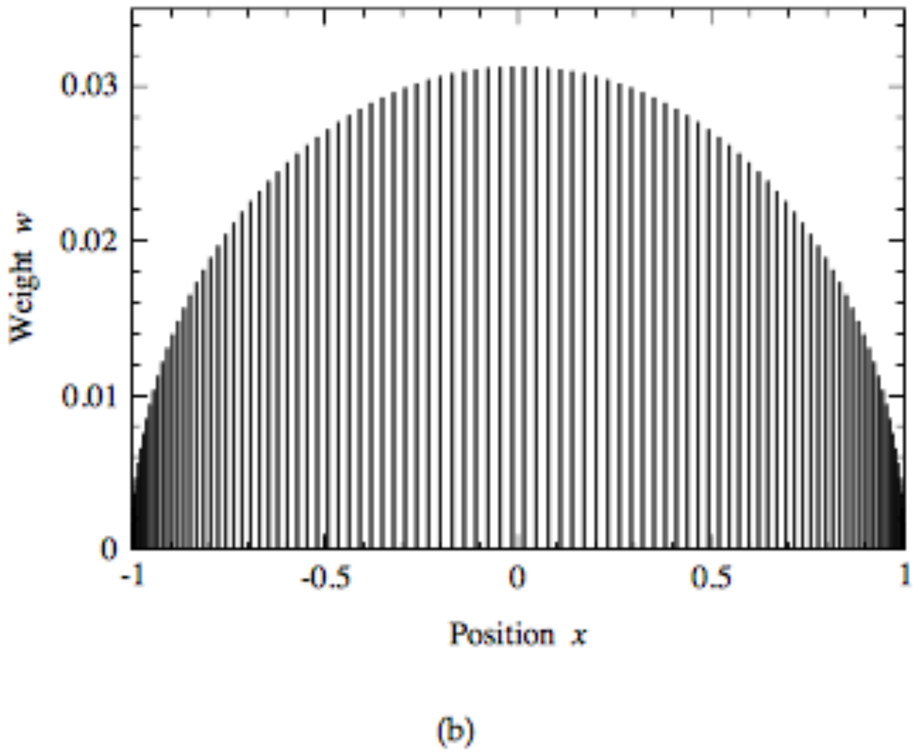


Figure 5.4: Sample points and weights for Gaussian quadrature. The positions and heights of the bars represent the sample points and their associated weights for Gaussian quadrature with (a) $N = 10$ and (b) $N = 100$.

$$\boxed{\int_a^b f(x)dx \approx \sum_{k=1}^{N} \mathrm{w}_k f(x_k)}$$

Gaussian Quadrature

- Text goes into details of how to find weights and sample points. Ends up sample points are chosen to coincide with zeros of Nth Legendre polynomial. (see appendix C)
- Don't write your own program to find sample points and weights. Usually use given subroutines.
- E.g. you have **gaussxw.py** and **gaussxwab.py**.

integration limits are -1 to 1

integration limits are a to b

- Why use gaussian quadrature: You can use many FEWER sample points to get the same level of accuracy as a N-C formula.

Gaussian Quadrature

- Text goes into details of how to find weights and sample points. Ends up sample points are chosen to coincide with zeros of Nth Legendre polynomial. (see appendix C)

- Don't write your own program to find sample points and weights. Usually use given subroutines.

- E.g. you have **gaussxw.py** and **gaussxwab.py**.

integration limits are -1 to 1

integration limits are a to b

- Why use gaussian quadrature: You can use many FEWER sample points to get the same level of accuracy as a N-C formula.

The calculation of weights and points is expensive so use gaussxw if you are going to change the limits of integration repeatedly.

Errors in Gaussian Quadrature

- complicated error formula, but in general: approximation error improves by a factor c/N^2 when you increase # of sample points by 1!!!

- e.g. going form N=10 to N=11 sample points improves your estimate by a factor of ~100. $\rightarrow$ converge very quickly to true value of the integral.

- Difficulties with gaussian quadrature:
  - only works well if function is reasonably smooth (since sample points are farther apart)

Gaussian Quadrature Example

- Integrate the function f(x) = x^4 + sin(x^2) from x= -1 to x=1 using Gaussian Quadrature. From Wolfram Alpha:

$$\int_{-1}^{1}(x^4+\sin(x^2))\,dx = \sqrt{2\,\pi}\;S\left(\sqrt{\frac{2}{\pi}}\right)+\frac{2}{5} \approx 1.02054$$

- Outline of procedure is always similar

- Lets come up with pseudocode…

Gaussian Quadrature Example

- Here is where we will write the pseudocode during class.

Gaussian Quadrature Example

- Now lets try and write the real code…
- open python…

Numerical Derivatives

- In a way, simpler than numerical integration!
  - But error characteristics are tricky.

- Strongly based on Taylor series approximations.

- Use Taylor series approximations to estimate errors

(1) Forward d

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

(2) Backwarc

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

Numerical Derivatives: Errors

- Use Taylor series to find error in these approximations:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\xi)$$

- Isolate for f ‘(x):

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(\xi)$$

- So we see that the error is first-order in h. (same is true for backward difference method).

Central Differences

- Can use Taylor series relationships to find sneaky improvements to finite difference schemes.

- E.g.: central difference method:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

- Notice it still only involves subtracting 2 points, its just that the location of the 2 points is different.

- Error:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_1) \\ f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(\xi_2) \end{aligned}$$

Numerical Derivatives

- Isolate for f'(x):

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12}[f'''(\xi_1) + f'''(\xi_2)]$$

- So we see that this formula is accurate to 2nd order in h.
- Can get higher order methods by including more points (see table 5.1 on page 196).
- Might have to do different things near the boundaries.
- Partial derivatives: similar techniques
- Higher order derivatives (e.g. f'') similar techniques.

The Return of Roundoff Error

- Let’s take another look at this formula:

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(\xi)$$

- What happens when we consider roundoff error?

- Well, each of the terms $f(x+h)$ and $f(x)$ have error ~$C|f(x)|$. And their difference will have approximate error $2Cf(x)$ (“worst case” error).

- So in fact there are two sources of error and this leads to (5.91)

$$\epsilon = \frac{2C\,|f(x)|}{h} + \tfrac{1}{2}h\,|f''(x)|. \qquad (5.91)$$

The Return of Roundoff Error

- So increasing N could increase the error.
- The error has a minimum at

$$h = \sqrt{4C\left|\frac{f(x)}{f''(x)}\right|}. \tag{5.93}$$

suggesting $N \sim O(10^{8})$. In this case, the error is $N \sim O(10^{-8})$, from

$$\epsilon = h\,|f''(x)| = \sqrt{4C\,|f(x)f''(x)|}. \tag{5.94}$$

- There are two points: there's a limit to the improvement you can obtain by going to finer resolution, and the precision expected on differentiation is orders of magnitude less than that of other operations we have discussed.

Solving Linear Systems

- In linear algebra courses you learn to solve linear systems of the form

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{v}$$

  using Gaussian elimination.
- This works pretty well in many cases. Let's do an example based on Newman's `gausselim.py`, for

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, v = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$$

Solving Linear Systems

- In linear algebra courses you learn to solve linear systems of the form

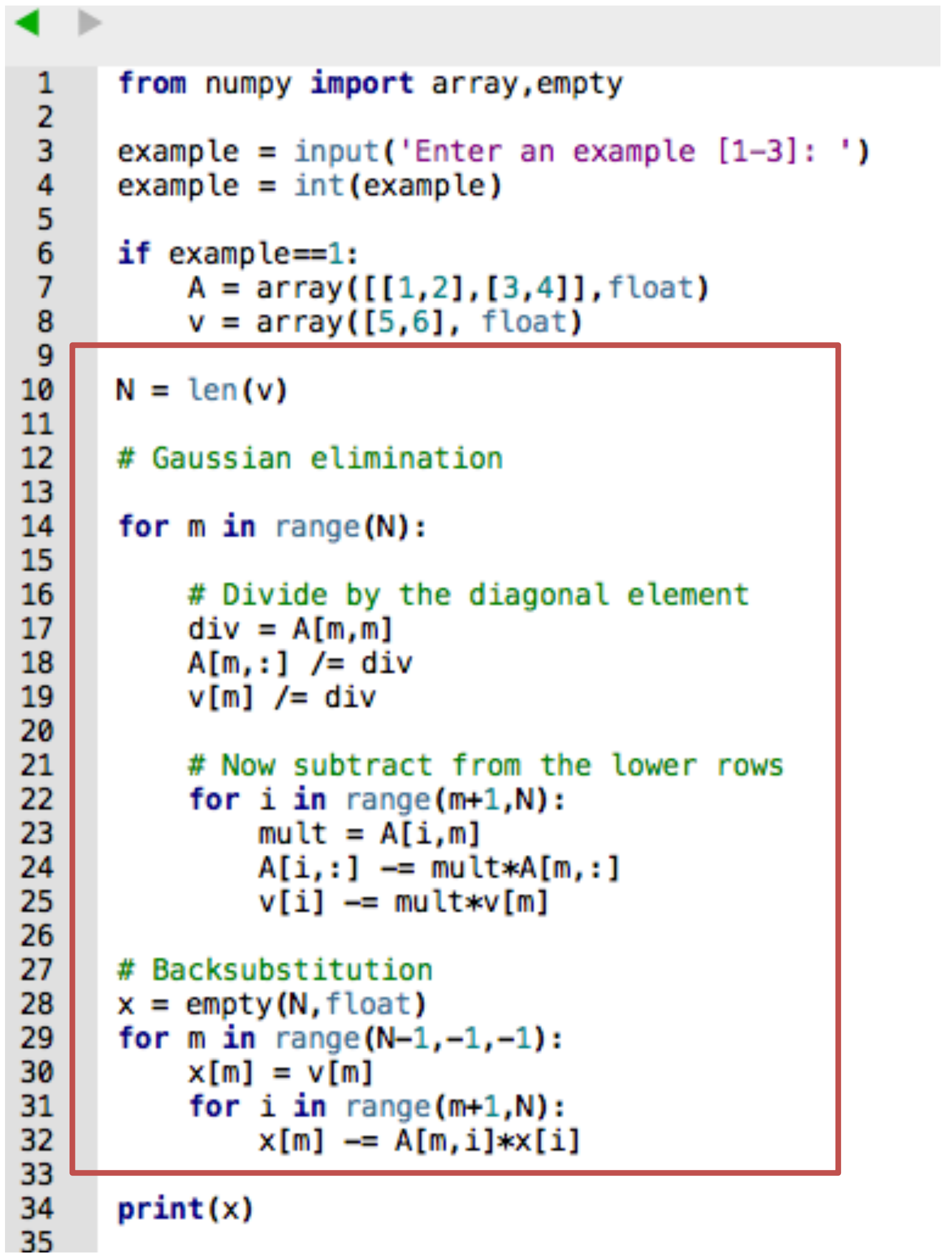
$$\mathbf{A}\boldsymbol{x} = \boldsymbol{v}$$

 using Gaussian elimination.

- This works pretty well in many cases. Let’s do an example based on Newman’s `gausselim.py`, for

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, v = \begin{pmatrix} 5 \\ 6 \end{pmatrix} \Rightarrow x = \begin{pmatrix} -4.0 \\ 4.5 \end{pmatrix}$$

Newman's approach



```
from numpy import array,empty

example = input('Enter an example [1-3]: ')
example = int(example)

if example==1:
    A = array([[1,2],[3,4]],float)
    v = array([5,6], float)

N = len(v)

# Gaussian elimination

for m in range(N):

    # Divide by the diagonal element
    div = A[m,m]
    A[m,:] /= div
    v[m] /= div

    # Now subtract from the lower rows
    for i in range(m+1,N):
        mult = A[i,m]
        A[i,:] -= mult*A[m,:]
        v[i] -= mult*v[m]

# Backsubstitution
x = empty(N,float)
for m in range(N-1,-1,-1):
    x[m] = v[m]
    for i in range(m+1,N):
        x[m] -= A[m,i]*x[i]

print(x)

```

The Problem with Gaussian Elimination

- But it’s easy to come up with examples that don’t work so well.
- The example below is a valid system but the original code will “break”.

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The Problem with Gaussian Elimination

- But it’s easy to come up with examples that don’t work so well.
- The example below is a valid system but the original code will “break”.

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow x \approx \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

The Problem with Gaussian Elimination

- Sample output from `gausselim-examples.py`

```
A is  [[  1.00000000e-20   1.00000000e+00] [
1.00000000e+00   1.00000000e+00]]
v is  [ 1.  0.]
x from linalg.solve is [-1.  1.]
but x from gaussian elimination is...[ 0.  1.]
```

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow x \approx \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

The Problem with Gaussian Elimination

- But it's easy to come up with examples that don't work so well.
- The example below is a valid system but the original code will "break".
- For any small number in the upper left hand corner, we'll tend to get inaccuracies.
- So we use pivoting (row swapping).

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow x \approx \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Partial Pivoting

- Sample output

```
A is  [[  1.00000000e-20   1.00000000e+00] [
1.00000000e+00   1.00000000e+00]]
v is  [ 1.  0.]
x from linalg.solve is [-1.  1.]
and x from partial pivoting is...[-1.  1.]
```

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow x \approx \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Summary & Status

- ☑Covered some python basics and pseudocoding …
- ☑Finished our first lab and pre-lab 02
- ☑Lecture02/Lab02/Tutorial 02: Programming tips, roundoff error, numerical integration.
- ❑Lecture03/Lab03/Tutorial03: Gaussian quadrature, numerical integration and differentiation.
  - Numerical integration, partial differentiation
  - First look at solving linear systems.
  - Tutorial: using `numpy` arrays, assign and discuss class project.