In [1]:

```
%pylab inline
%config InlineBackend.figure_format = 'retina'
from ipywidgets import interact
```

```
Populating the interactive namespace from numpy and matplotlib
```

Exam Rules

1. You can use your own notes and any class materials posted eClass, including Jupyter notebooks and anything else posted or linked to in eClass (however, see 4 below)
2. You cannot use any external internet sources (e.g., Google, Wikipedia, StackExchange, etc)
3. You can use your own books as long as they are paper or PDF
4. You cannot use any sort of computer search functionality (internet or otherwise)
5. You must work alone for this exam
6. You may use any theorems proved in class
7. Your answers do not need to be rigorous proofs, but try to be precise

You must turn in your exam before midnight on Saturday (Oct 8th)

(Even if you start it during the day on Saturday.)

Q1

Let $x_0, \ldots, x_n \in [a,b]$, and let f be sufficiently smooth on $[a,b]$ (i.e., $f \in C^{n+1}[a,b]$). Suppose that $\hat{p}(x)$ is a polynomial of degree at most n satisfying $|f(x_i) - \hat{p}(x_i)| < \epsilon$ for $i = 0, \ldots, n$. Show that

$$|f(x) - \hat{p}(x)| \leq \frac{\|f^{(n+1)}\|_\infty}{(n+1)!} \prod_{i=0}^{n} |x - x_i| + \kappa\epsilon$$

for all $x \in [a,b]$, in which κ is the condition number (Lebesgue constant).

Solution:

Let $p(x)$ be the polynomial of degree at most n that interpolates f at x_i, $i = 0, \ldots n$. We know this polynomial exists and is unique by the existence uniqueness theorem. Since $p(x_i) = f(x_i)$ it follows that

$$|f(x_i) - \hat{p}(x_i)| = |f(x_i) - p(x_i) - (\hat{p}(x_i) - p(x_i))| = |\hat{p}(x_i) - p(x_i)| < \epsilon.$$

From the above inequality, we can use the stability theorem given in lecture to obtain the bound

$$|\hat{p}(x) - p(x)| < \kappa\epsilon,$$

for all $x \in [a,b]$.

From the interpolation accuracy theorem given in lecture, we have that

$$|f(x) - p(x)| \leq \frac{\|f^{(n+1)}\|_\infty}{(n+1)!} \prod_{i=0}^{n} |x - x_i|,$$

for all $x \in [a,b]$.

Finally, from the triangle inequality we have

$$|f(x) - \hat{p}(x)| = |f(x) + p(x) - (\hat{p}(x) - p(x))| < |\hat{p}(x) - p(x)| + |f(x) - p(x)|.$$

Combining these three inequalities yields the desired result.

In [ ]:

Q2

Consider the series

$$S_N = \sum_{n=1}^{N} \frac{1}{n}.$$

A

Show that the limit

$$\lim_{N\to\infty} S_N = \infty.$$

Solution:

It suffices to apply an integral test (which also helps us with part D) since the terms $1/n$ are monotonically decreasing. We can bound the partial sums from below with

$$f(N) = \int_1^N \frac{1}{x} dx < \sum_{n=1}^{N} \frac{1}{n}.$$

Since $f(x) = \log(x) \to \infty$ as $x \to \infty$, the series diverges.

B

Explain why summing the series in finite precision floating point arithmetic converges after a finite number of terms; i.e.,

$$\sum_{n=1}^{\infty} \frac{1}{\text{fl}(n)} = L < \infty.$$

Solution:

The set of floating point numbers (either 32bit and 64bit) has a finite number of elements. It follows that there is a largest floating point number. Any real number that is larger than the largest floating point number is given the value of `inf` (called overflow). The corresponding value of `1/inf` is considered to be zero. Hence, there exists an $N > 0$ such that $\text{fl}(\frac{1}{\text{fl}(n)}) = 0$ for all $n > N$. The sequence of partial sums is non decreasing. We must therefore have a finite limit as $n \to \infty$ since $S_n \leq S_N$ for all n.

C

Try to predict the number of terms N after which the partial sum S_N will cease to change for

1. 32 bit floating point numbers
2. 64 bit floating point numbers

Solution:

The question only asks you to **try** to predict the value. I gave full marks for any reasonable guess based on facts about the floating point number system. An allowable answer can be based on the largest floating point number even though this turns out to be a bad guess. Since the true cause of convergence is rounding error (i.e., from $\mathrm{fl}(S_{n-1} + \frac{1}{n})$) a better guess would be $1/\varepsilon$, where ε is the rounding unit. The rounding unit is approximately 10^{-8} and 10^{-16} for `float32` and `float64`, respectively.

D

Confirm your prediction from part C numerically.

Hint: This should just require a simple `for` *loop and an* `if` *statement. Use the* `float32()` *function to compute the sum in 32 bit float (make sure you do this correctly). For float64, you might not want to compute every single term in the sum, unless you are happy to potentially wait a long time.*

Answer:

In part C, you might have formulated a bad guess based on an incorrect reason for convergence of the sum. The cause of numerical convergence of the sum is due to rounding error from $\mathrm{fl}(\mathrm{fl}(S_{n-1}) + \mathrm{fl}(\frac{1}{\mathrm{fl}(n)}))$. As discussed in an example during lecture, if we add a very large floating point number L and a much smaller floating point number s, we have $\mathrm{fl}(L + s) = L$. This is due to the finite number of digits in the mantissa, which determines the rounding unit ε.

In [2]:

```
## The float32 sum can be computed directly
Sn = float32(0)
for n in arange(1, 10**7):
    y = Sn + float32(1)/float32(n)
    if absolute(Sn - y) == 0:
        break
    Sn = y
print(n)
```

```
2097152
```

float64 case

I am not taking off any points if you couldn't think of a way to "short cut" the sum for float64. Since S_n grows like $\log(n)$, we can approximate. Let us assume that $S_n \sim C + \log(n)$, $n \to \infty$, for some constant C.

We will first approximate the constant

Note: the constant probably does not affect the answer, but we will include it anyway since it is easy to approximate. By the way, C should be Euler's constant γ.

In [3]:

```
S0 = 0.
for n0 in arange(1, 10**7):
    S0 += 1./n0

## we will restart the sum at a very large number n1
## S0 ~ C + log(n0) =>  C ~ S0 - log(n0)
C = S0 - log(n0)
print(C)
```

0.577215714898955

In [4]:

```
guess = 1/finfo(float64).eps/32
print(guess)
```

140737488355328.0

In [5]:

```
## this should yield an approximation that has around 8 correct digits
n1 = around(guess)
n = n1
for _ in arange(1e8):
    n += 1e7
    Sn = C + log(n) ## approximate
    y = Sn + 1./n
    if absolute(Sn - y) == 0:
        break

print(n)
```

281474978355328.0

In [6]:

```
## this should yield the (possibly exact) answer
n2 = n - 1e7
n = n2
for _ in arange(1e7):
    n += 1
    Sn = C + log(n) ## approximate
    y = Sn + 1./n
    if absolute(Sn - y) == 0:
        break

print(n)
```

281474976710657.0

Q3

Suppose we are constructing a numerical library for early computers. We wan to devise an efficient algorithm for computing square roots. We can only use the basic arithmetic operations ($*, /, +, -$). Newton's method can be used to find the square root of a positive number $a > 0$.

A

Formulate the computation of $x = \sqrt{a}$ as a root finding problem. Derive a fixed point iteration using Newton's method. (Your iteration should only use the basic arithmetic operations $*, /, +, -$.) Show that in this case, we can expect the Newton iteration to converges quadratically for any $a > 0$.

Solution:

Let $f(x) = x^2 - a$. Clearly $f(x) = 0$ implies that $x = \pm\sqrt{a}$, and we can choose the positive solution. Newton's method is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

In our case, we have that $f'(x) = 2x$. Hence,

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n}.$$

As shown in class, Newton's method converges quadratically given a sufficiently smooth function f and $f'(\sqrt{a}) \neq 0$. Clearly f is smooth. Since $a > 0$ by assumption, we have $f'(\sqrt{a}) = 2\sqrt{a} > 0$.

B

Suppose we have processor with limited functionality. It can compute $+, -, *$, but cannot compute general division /. Instead, it can only half a number (i.e., divide by 2). Follow a similar procedure as in part A to devise a fixed point iteration for computing $x = 1/\sqrt{a}$. It is then simple to compute $\sqrt{a}$ by multiplying $1/\sqrt{a}$ by a. (Your iteration should only use $+, -, *$ and division by 2.) Show that the iteration converges quadratically for any $a > 0$.

Solution:

Let $f(x) = \frac{1}{x^2} - a$. We have that $f(x) = 0$ implies that $x = \pm\frac{1}{\sqrt{a}}$. We have that $f'(x) = -\frac{2}{x^3}$. Newton's method is then

$$x_{n+1} = x_n - \frac{\frac{1}{x_n^2} - a}{-\frac{2}{x_n^3}} = x_n + \frac{x_n - ax_n^3}{2}.$$

Note that the above uses only the allowed operations operations, $+, -, *$ and division by 2. Our function is smooth in a neighborhood of the root, and since $a > 0$ by assumption, we have that $f'(\frac{1}{\sqrt{a}}) > 0$. Hence the iteration will converge quadratically.

In [ ]: