Relative cost of matrix operations

In [1]:

```
#keep
import numpy as np
import scipy.linalg as spla
import scipy as sp

import matplotlib.pyplot as pt

from time import time
%matplotlib inline

np.alterdot()
```

In [2]:

```
#keep
n_values = (10**np.linspace(1, 3.75, 15)).astype(np.int32)
n_values
```

Out[2]:

```
array([  10,   15,   24,   38,   61,   95,  150,  237,  372,  585,
921,
       1447, 2275, 3577, 5623], dtype=int32)
```

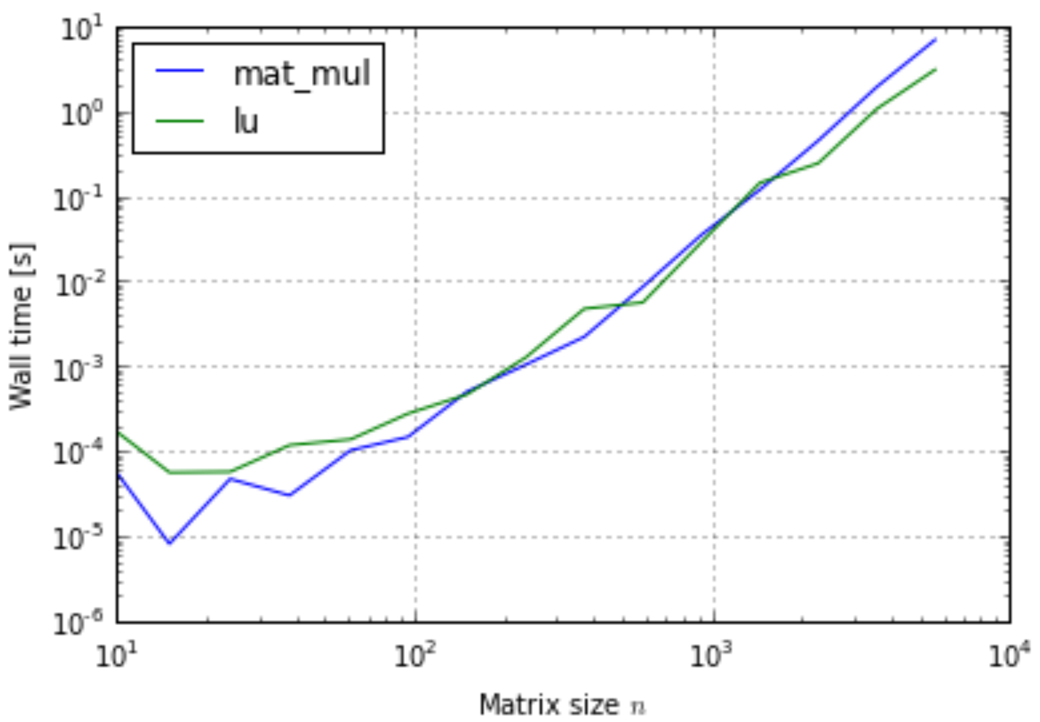
In [3]:

```
#keep

def mat_mul(A):
    return A.dot(A)

for name, f in [
        ("mat_mul", mat_mul),
        ("lu", spla.lu_factor),
        ]:

    times = []
    print("----->", name)

    for n in n_values:
        print(n)

        A = np.random.randn(n, n)

        start_time = time()
        f(A)
        times.append(time() - start_time)

    pt.plot(n_values, times, label=name)

pt.grid()
pt.legend(loc="best")
pt.xlabel("Matrix size $n$")
pt.ylabel("Wall time [s]")
```

```
-----> mat_mul
10
15
24
38
61
95
150
237
372
585
921
1447
2275
3577
5623
-----> lu
10
15
24
38
61
95
150
237
372
585
921
1447
2275
3577
5623
```

Out[3]:

```
<matplotlib.text.Text at 0x10dc5acc0>
```



- The faster algorithms make the slower ones look bad. But... it's all relative.
- Is there a better way of plotting this?
- Can we see the asymptotic cost ($O(n^3)$) of these algorithms from the plot?

In [ ]:

```
pt.show()
```

In [ ]:

```

```