Preconditioner On GMRes

January 20, 2016

In [23]:
```
# Libraries
import numpy as np
import time
import scipy.sparse.linalg as spy
import warnings
import matplotlib.pyplot as plt
%matplotlib inline

warnings.filterwarnings('ignore')


def pos_def(n,x=5):
    A = np.random.rand(n,n)
    return (A+A.T) + x*np.eye(n)

alpha = 0.05

def graphics(n, at=1e-6, Nf=128):
    it_1 = []
    it_2 = []
    it_3 = []
    er1 = []
    er2 = []
    er3 = []
    t1 = []
    t2 = []
    t3 = []
    x1 = []
    x2 = []
    x3 = []
    x = range(10, n+10,10)
    for i in range(10,n+10,10):
        A = np.array(np.fromfile("matrices/m{0}".format(i))).reshape((i,i))
        x_sol = np.floor(np.random.random(i) * 100)
        b = np.dot(A, x_sol)

        start = time.time()
        r1 = gmres(A,b)
        t1.append(time.time() - start)

        start = time.time()
        r2 = prec_gmres(A,b,alpha)
        t2.append(time.time() - start)
```

```
    start = time.time()
    r3 = gmres(A, b,prec=True,alpha=alpha,aux_tol=at,N=Nf)
    t3.append(time.time() - start)

    it_1.append(r1[1])
    it_2.append(r2[1])
    it_3.append(r3[1])

    er1.append(np.linalg.norm(np.dot(A,r1[0]) - b)/np.linalg.norm(b))
    er2.append(np.linalg.norm(np.dot(A,r2[0]) - b)/np.linalg.norm(b))
    er3.append(np.linalg.norm(np.dot(A,r3[0]) - b)/np.linalg.norm(b))

    x1.append(np.linalg.norm(r1[0] - x_sol)/np.linalg.norm(x_sol))
    x2.append(np.linalg.norm(r2[0] - x_sol)/np.linalg.norm(x_sol))
    x3.append(np.linalg.norm(r3[0] - x_sol)/np.linalg.norm(x_sol))

fig = plt.figure()
ax = fig.add_subplot(111)
l1, l2, l3 = ax.plot(x,it_1, 'r:^', x,it_2, 'g:^', x, it_3,'b:^')
ax.set_xlim(0, n)
ax.set_ylim(0, np.max(it_1)+3)
fig.legend((l1, l2, l3), ('GMRes', 'Prec GMRes', 'Cauchy GMRes'), 'upper right')
plt.xlabel('Matrix dimension')
plt.ylabel('Iterations')
plt.title('Iterations comparison')
plt.show()


fig = plt.figure()
ax = fig.add_subplot(111)
l1, l2, l3 = ax.plot(x,er1, 'r:^', x,er2, 'g:^', x, er3,'b:^')
ax.set_yscale("log")
ax.set_xlim(0, n)
ax.set_ylim(0, (np.max(er1)+np.min(er1))/2+ np.max(er1))
fig.legend((l1, l2, l3), ('GMRes', 'Prec GMRes', 'Cauchy GMRes'), 'upper right')
plt.xlabel('Matrix dimension')
plt.ylabel('Errors')
plt.title('Errors comparison')
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
l1, l2, l3 = ax.plot(x,t1, 'r:^', x,t2, 'g:^', x, t3,'b:^')
ax.set_yscale("log")
ax.set_xlim(0, n)
ax.set_ylim(0, np.max(t3))
fig.legend((l1, l2, l3), ('GMRes', 'Prec GMRes', 'Cauchy GMRes'), 'upper right')
plt.xlabel('Matrix dimension')
plt.ylabel('Time')
plt.title('Time comparison')
plt.show()
```

```
    fig = plt.figure()
    ax = fig.add_subplot(111)
    l1, l2, l3 = ax.plot(x,x1, 'r:^', x,x2, 'g:^', x, x3,'b:^')
    ax.set_yscale("log")
    ax.set_xlim(0, n)
    ax.set_ylim(0, np.max(x1)+3)
    fig.legend((l1, l2, l3), ('GMRes', 'Prec GMRes', 'Cauchy GMRes'), 'upper right')
    plt.xlabel('Matrix dimension')
    plt.ylabel('Forward Error')
    plt.title('Forward Errors comparison')
    plt.show()

    return

#Lambda Relationship
def plot_lambdas(A):
    lambdas, v = np.linalg.eig(A)
    func = np.vectorize(lambda x: (1. - (alpha/(2*x))**(50+1))/(x - alpha/2.))

    x = lambdas
    y1 = lambdas*1./(lambdas + alpha)
    y2 = func(lambdas + alpha)*lambdas

    fig = plt.figure()
    ax = fig.add_subplot(111)
    l1, l2 = ax.plot(x, y1, 'r^', x,y2, 'g^')
    ax.set_xlim(-10, np.max(lambdas)+2)
    ax.set_ylim(-10, np.max([np.max(y1), np.max(y2)])+2)
    fig.legend((l1, l2), ('Prec GMRes', 'Cauchy GMRes'), 'upper right')
    plt.xlabel('Lambdas')
    plt.ylabel('f(lambda + alpha)')
    plt.title('Preconditioner quality')
    plt.show()

#Accuracy vs Time Graphic
def accuracyTime(n):
    t1 = []
    t2 = []
    t3 = []

    tol = 10**(-1.0*np.array(range(n)))


    A = np.array(np.fromfile("matrices/m50").reshape((50,50)))
    x_sol = np.floor(np.random.random(50)* 100)
    b = np.dot(A, x_sol)


    for i in range(n):
        start = time.time()
        gmres(A, b,prec=True,alpha=alpha,aux_tol=10**(-i),N=4)
        t3.append(time.time() - start)
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
l3 = ax.plot(tol, t3,'b:^')
ax.set_xlim(-0.5, np.max(tol))


ax.set_ylim(0, np.max(t3)+np.max(t3)/2)
#ax.set_xscale('log')
fig.legend((l3), ('Cauchy GMRes'), 'upper right')
plt.xlabel('Tolerance')
plt.ylabel('Time')
plt.title('Tolerance v/s Time comparison')
plt.show()
```

In [24]:
```
# A better implementation of preconditioned GMRes without Cauchy integral (left)
def prec_gmres(A_0,b_0,alpha,tol=1e-6,left = True):
    it = b_0.shape[0]
    Q = np.zeros((b_0.shape[0], it+1))
    H = np.zeros((it+1,it))
    x0 = np.zeros((b_0.shape[0]))

    # copy
    A = np.copy(A_0)
    b = np.copy(b_0)
    M = A + alpha*np.identity(b.shape[0])

    # Left preconditioner -> A and b changes
    if left:
        b = gmres(M,b,tol=tol)[0]

    r = b - np.dot(A,x0)
    beta0 = np.linalg.norm(b)
    beta1 = np.linalg.norm(r)
    Q[:,0] = r/beta1

    for i in range(it):
        e = np.zeros((i+2))
        e[0] = 1

        if left:
            w = gmres(M, np.dot(A,Q[:,i]),tol=tol)[0]
        else:
            w = np.dot(A, gmres(M, Q[:,i],tol=tol)[0])

        for j in range(i+1):
            h = np.dot(Q[:,j],w)
            w -= h*Q[:,j]
            H[j,i] = h

        H[i+1,i] = np.linalg.norm(w)

        if H[i+1,i] != 0:
            Q[:,i+1] = w/H[i+1,i]
```

```
            y,_,_,_ = np.linalg.lstsq(H[:i+2,:i+1], beta1*e)
            residual = np.linalg.norm(np.dot(H[:i+2,:i+1],y) - beta1*e)

            if H[i+1,i] == 0 or residual/beta0 < tol:
                break

        x_tild = np.dot(Q[:,:i+1], y)
        if left:
            return x_tild, i+1
        else:
            return gmres(M, x_tild,tol=tol)[0],i+1
```

```
In [25]: def trapezoid2(myfun, N, a, b):
             x = np.linspace(0, b, N/2) # We want N bins, so N+1 points
             h = x[1]-x[0]
             xmiddle = x[1:-1]
             int_val = 0
             for i in xmiddle:
                 int_val += 2*myfun(i).real
             int_val = 2*myfun(a).real + 2*int_val + 2*myfun(0)# + myfun(b)
             return 0.5*h*int_val

         def z(t, c, r):
             return c + r*np.complex(np.cos(t), np.sin(t))

         def dz(t, r):
             return r*np.complex(np.cos(t), np.sin(t))

         def g(t,l,L,alpha,A, v,f, tol=1e-6):
             centro = (l + L)/2.
             radio = (L - l)/2.
             fz = f(z(t, centro, radio))
             dzz = dz(t, radio)
             p = fz*dzz
             # Separamos las matrices
             M = (centro + dzz - alpha)*np.identity(v.shape[0]) - A
             a = M.real
             b = M.imag
             al = v.real
             bet = v.imag

             # Construímos el nuevo sistema Hx = r
             H = np.zeros((a.shape[0]*2, a.shape[1]*2))
             H[:a.shape[0],:a.shape[1]] = a
             H[a.shape[0]:,a.shape[1]:] = a
             H[:a.shape[0],a.shape[1]:] = -b
             H[a.shape[0]:,:a.shape[1]] = b
             r = np.zeros((al.shape[0]*2))
             r[:al.shape[0]] = al
             r[al.shape[0]:] = bet

             sol = gmres(H,r,tol=tol)[0]
             gmr = sol[:(sol.shape[0]/2)] + 1j*sol[(sol.shape[0]/2):]
             return p*gmr
```

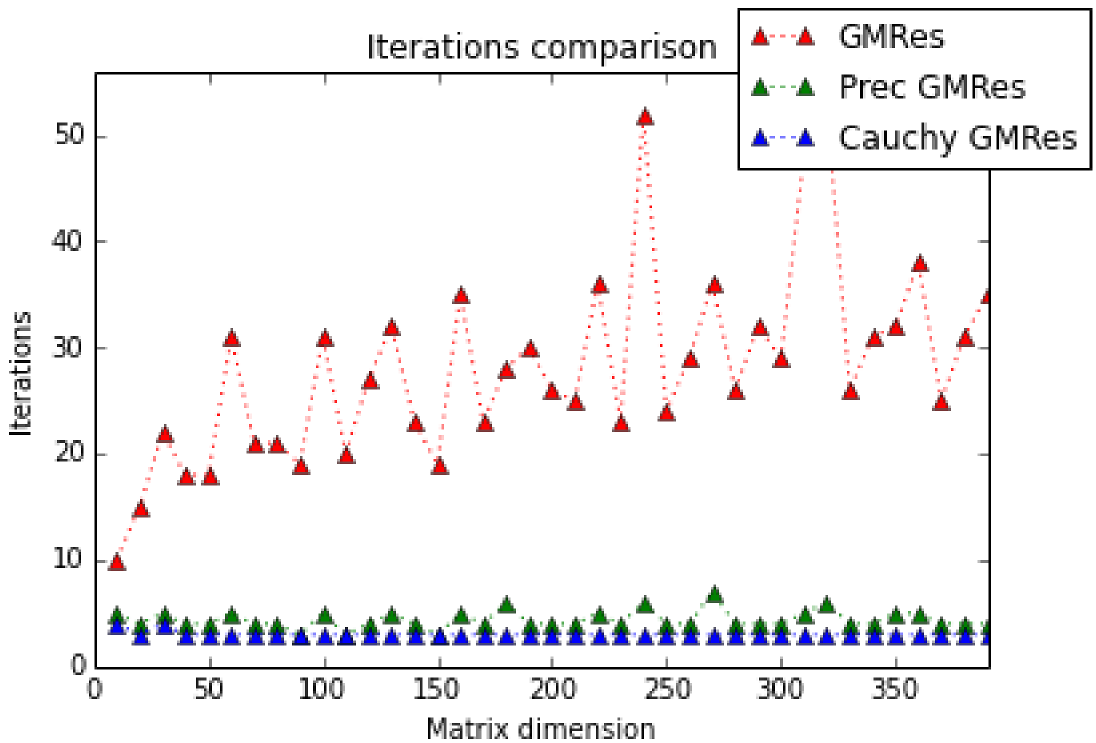
```
def cauchy_integral(l, L, alpha, A, v, Nf = 50,N=32,tol=1e-6):
    f = lambda x: (1. - (alpha/(2*x))**(Nf+1))/(x - alpha/2.)
    g1 = lambda t: g(t,l,L,alpha,A,v,f,tol=tol)
    val = trapezoid2(g1, N, -np.pi, np.pi) / (2.*np.pi)
    return val
```

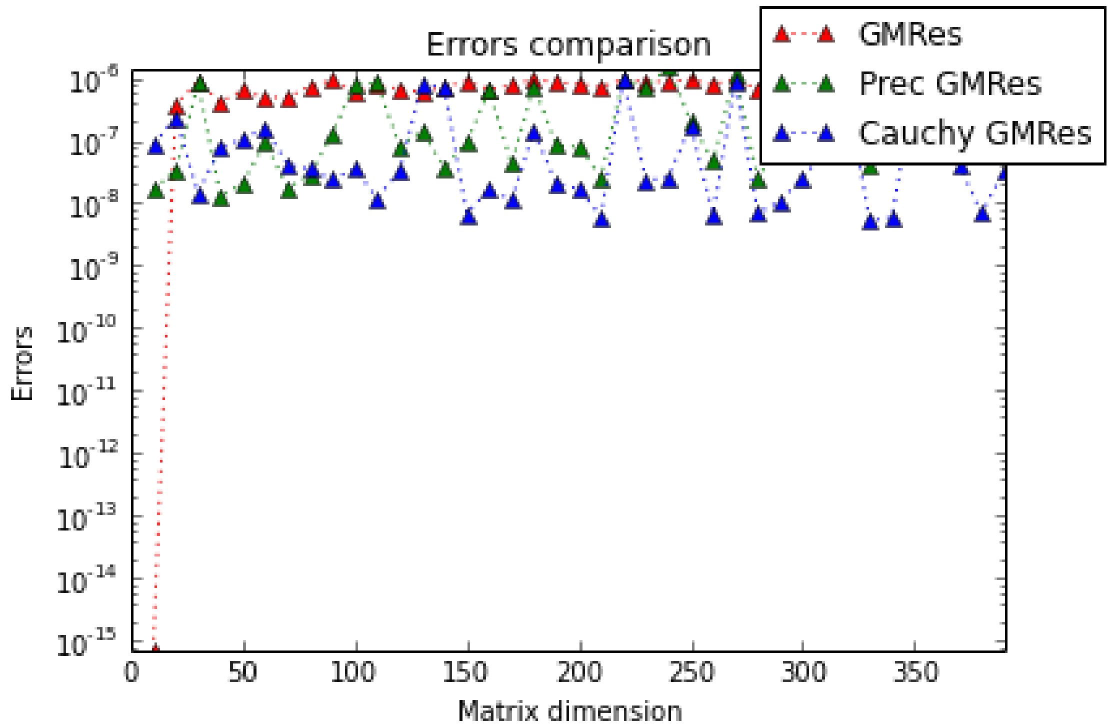
In [26]:

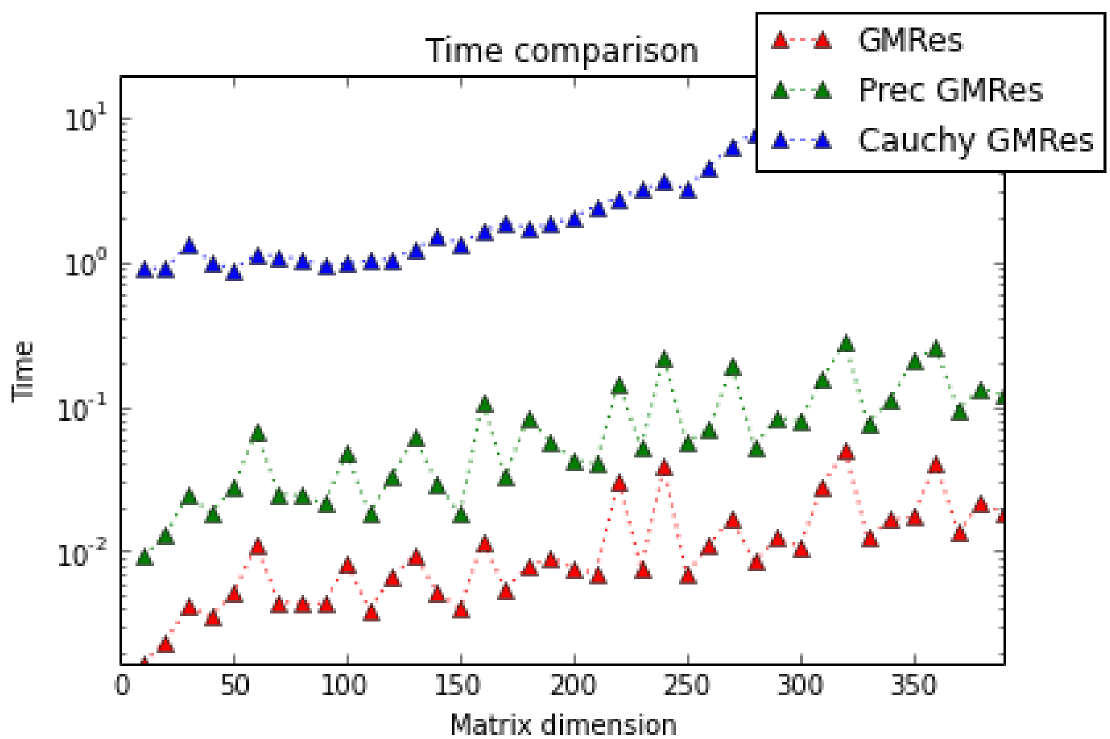
```
# GMRes using contour integral to compute the preconditioner
def gmres(A_0, b_0, tol=1e-6, prec=False, left=True, alpha=0.0,aux_tol=1e-6,N = 128):
    it = b_0.shape[0]
    Q = np.zeros((b_0.shape[0], it+1))
    H = np.zeros((it+1,it))
    x0 = np.zeros((b_0.shape[0]))

    A = np.copy(A_0)
    b = np.copy(b_0)

    if prec:
        l = alpha
        L = np.amax(np.sum(np.abs(A),axis=1))
        if left:
            b = cauchy_integral(l, L, alpha, A, b,tol=aux_tol, N=N)

    r = b
    beta0 = np.linalg.norm(b)
    beta1 = np.linalg.norm(r)
    Q[:,0] = r/beta1

    for i in range(it):
        e = np.zeros((i+2))
        e[0] = 1

        if prec:
            if left:
                w = cauchy_integral(l,L,alpha,A, np.dot(A,Q[:,i]),tol=aux_tol, N=N)
            else:
                w = np.dot(A, cauchy_integral(l, L, alpha, A, Q[:,i],tol=aux_tol, N=N))
        else:
            w = np.dot(A,Q[:,i])

        for j in range(i+1):
            h = np.dot(Q[:,j],w)
            w -= h*Q[:,j]
            H[j,i] = h

        H[i+1,i] = np.linalg.norm(w)

        if H[i+1,i] != 0:
            Q[:,i+1] = w/H[i+1,i]

        y,_,_,_ = np.linalg.lstsq(H[:i+2,:i+1], beta1*e)
        residual = np.linalg.norm(np.dot(H[:i+2,:i+1],y) - beta1*e)
```

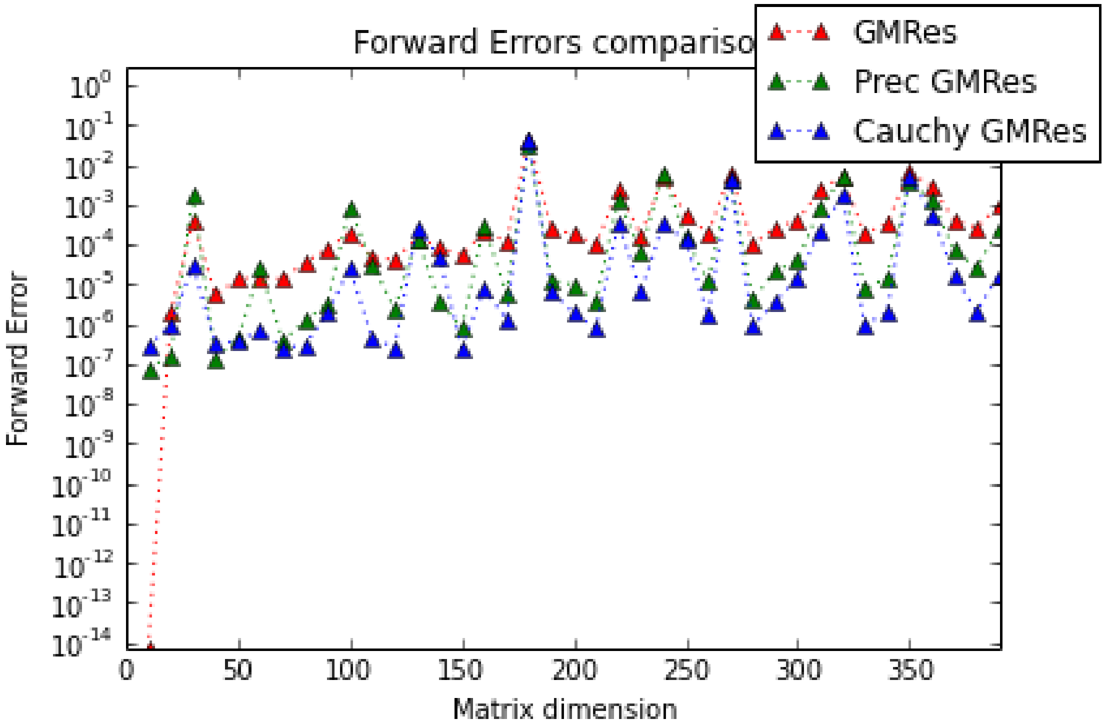
```
            if H[i+1,i] == 0 or residual/beta0 < tol:
                break

        x_tild = np.dot(Q[:,:i+1], y)
        if left or not prec:
            return x_tild,i+1
        else:
            return cauchy_integral(l, L, alpha, A, x_tild,tol=aux_tol, N=N),i+1
```

In [22]:
```
# Grafico original, N = 128, se integra en todos los puntos, tol = tol_aux = 1e-6
graphics(390)
```



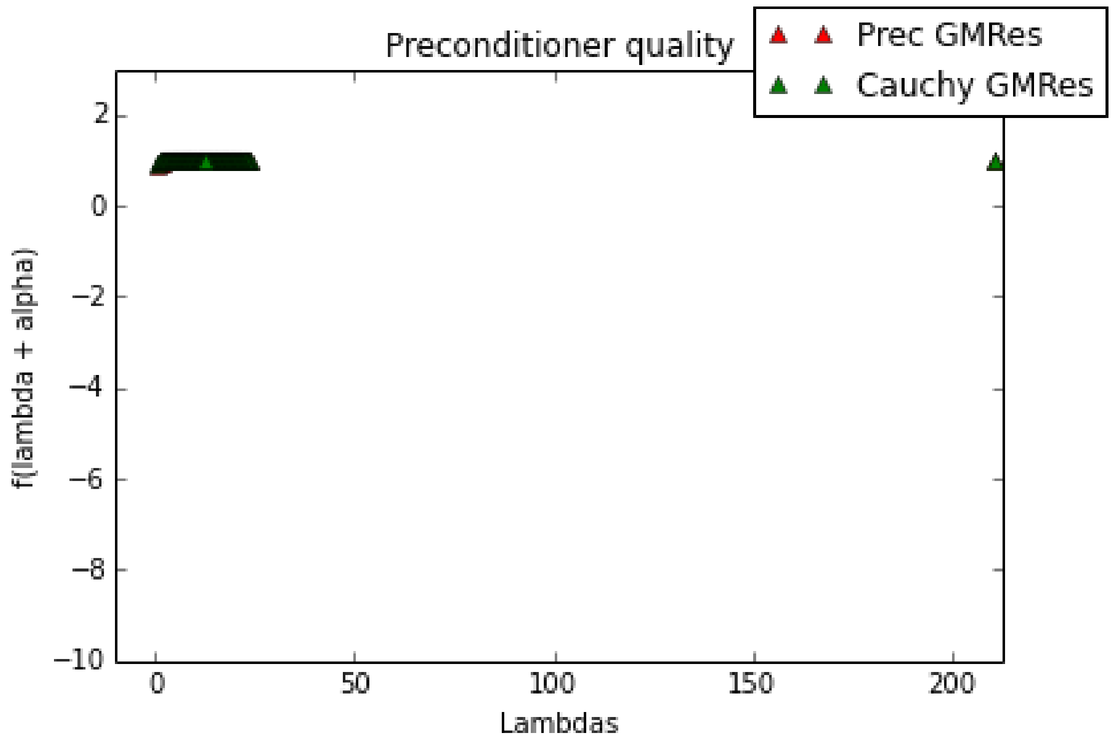

Errors comparison
GMRes
Prec GMRes
Cauchy GMRes
Errors
Matrix dimension


Time comparison
GMRes
Prec GMRes
Cauchy GMRes
Time
Matrix dimension



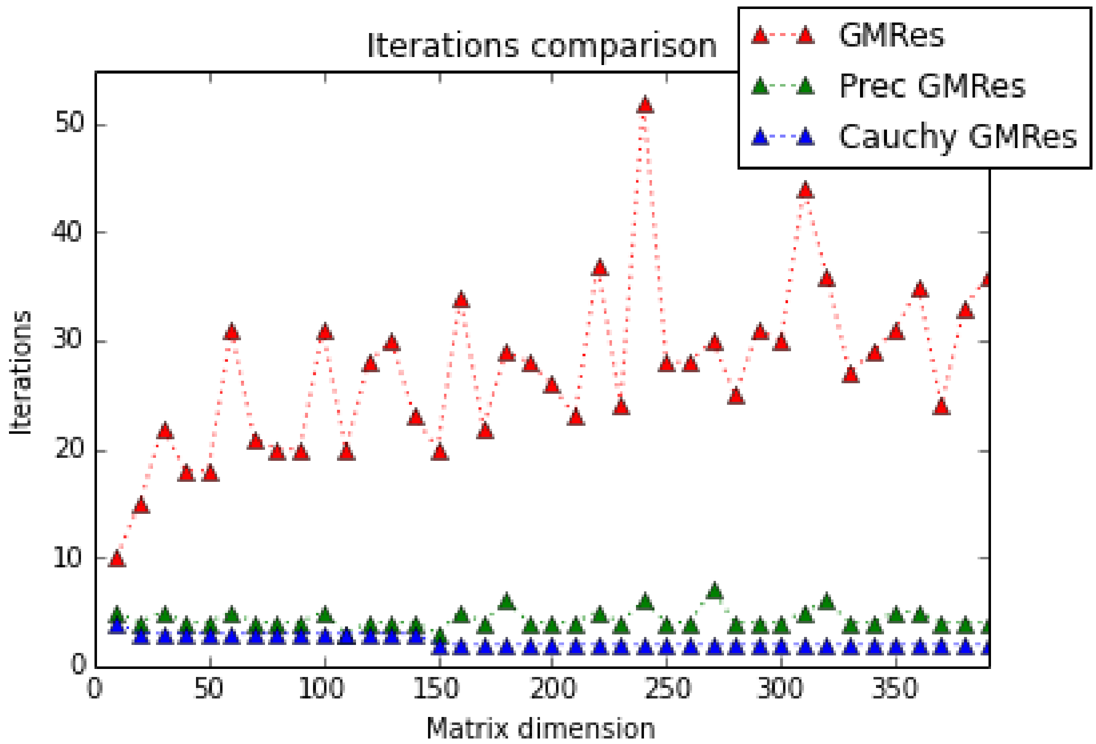
In [15]: # N = 128, se integra en pocos puntos cerca de la cota inferior, tol = tol_aux = 1e-6

```
for i in range(10,400,10):
    A = np.array(np.fromfile("matrices/m{0}".format(i))).reshape((i,i))

    if i % 200 == 0:
        plot_lambdas(A)
```

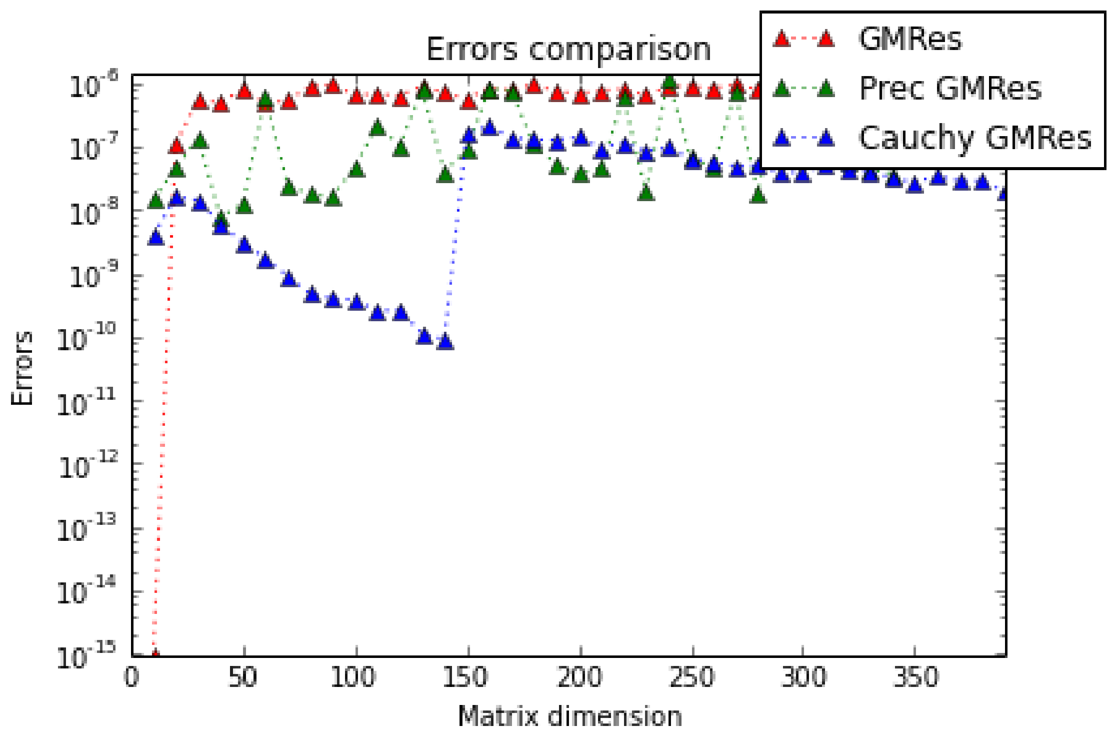



```
In [11]: print "N = 32 , a_tol= 1e-10"
         graphics(390, at = 1e-12, Nf=32)

         print "N = 16 , a_tol= 1e-12"
         graphics(390, at=1e-12, Nf=16)

         print "N = 8 , a_tol= 1e-10"
         graphics(390, at=1e-12, Nf=8)

         print "N = 4 , a_tol= 1e-12"
         graphics(390, at=1e-12, Nf=4)

         print "N = 16 , a_tol= 1e-6"
         graphics(390, at=1e-6, Nf=16)
```

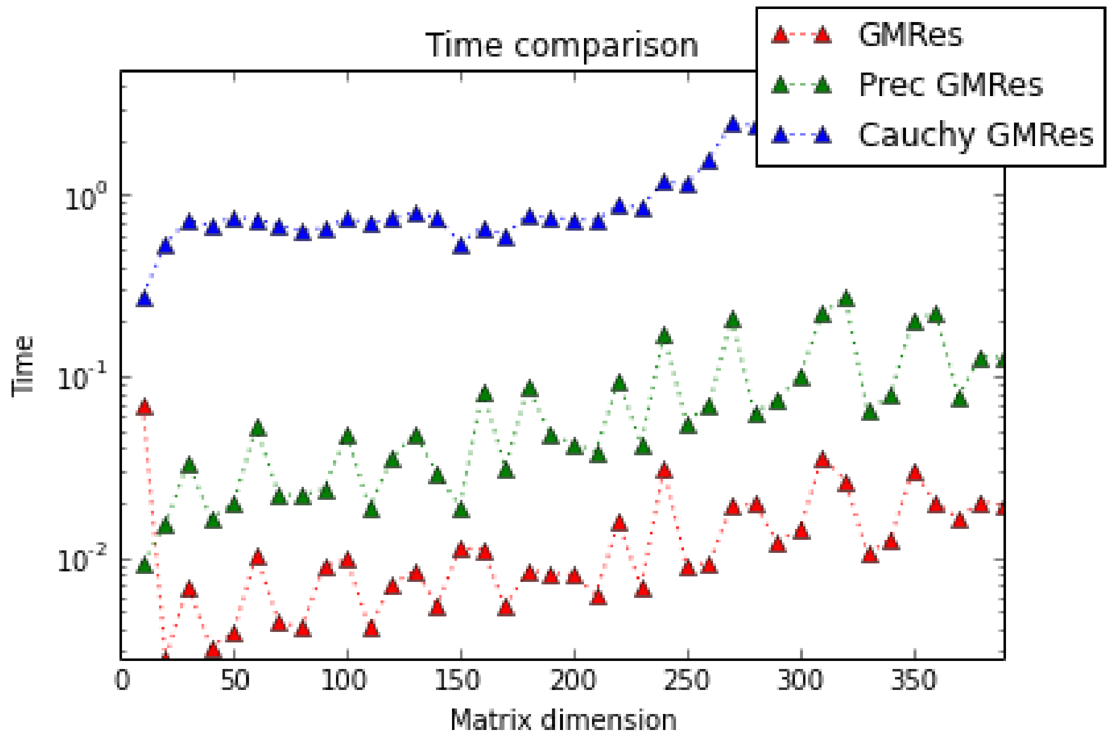
```
N = 32 , a_tol= 1e-10
```

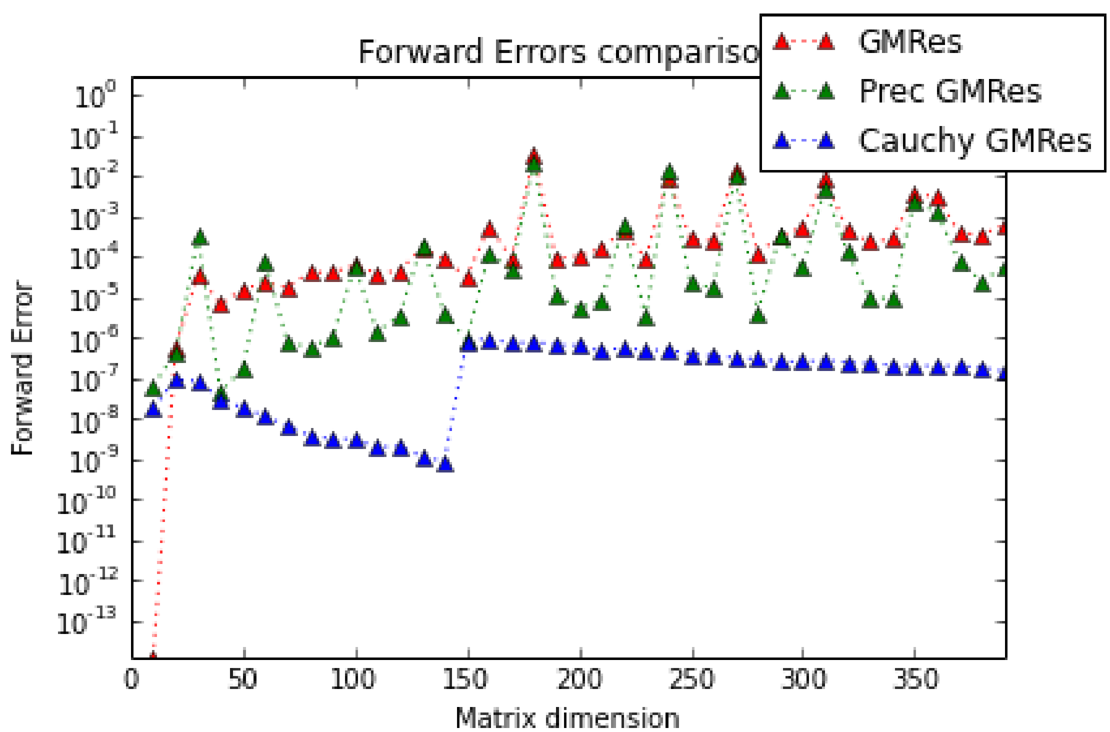


Iterations comparison
GMRes
Prec GMRes
Cauchy GMRes
Iterations
Matrix dimension

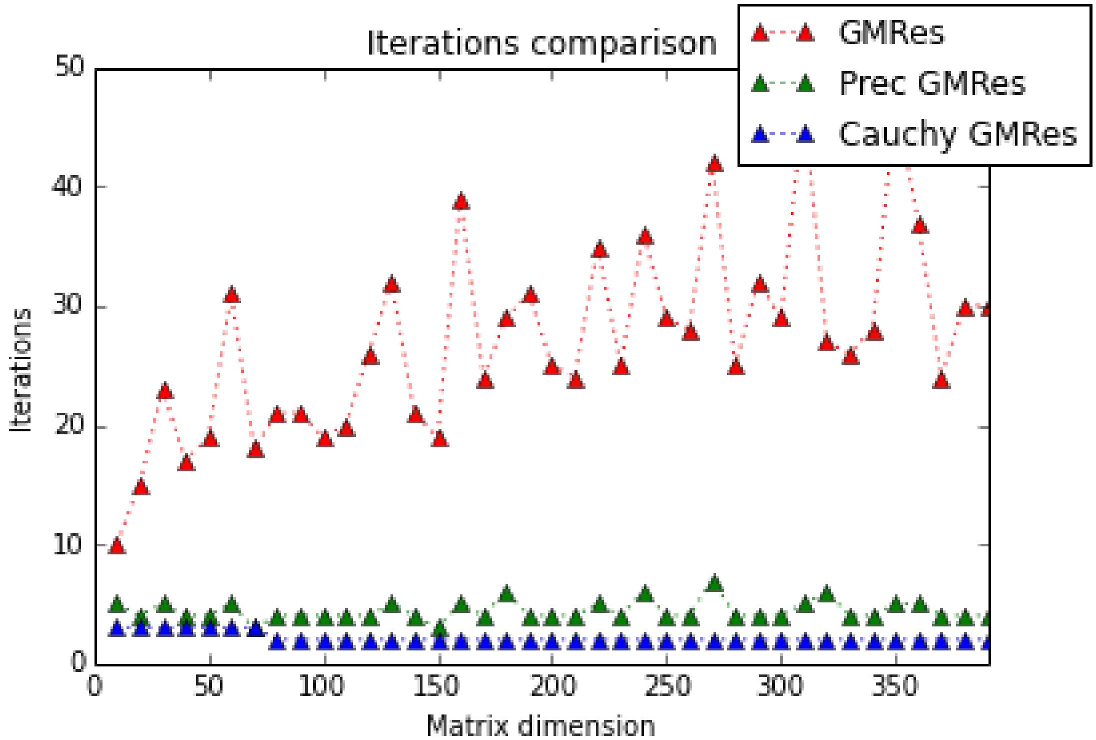


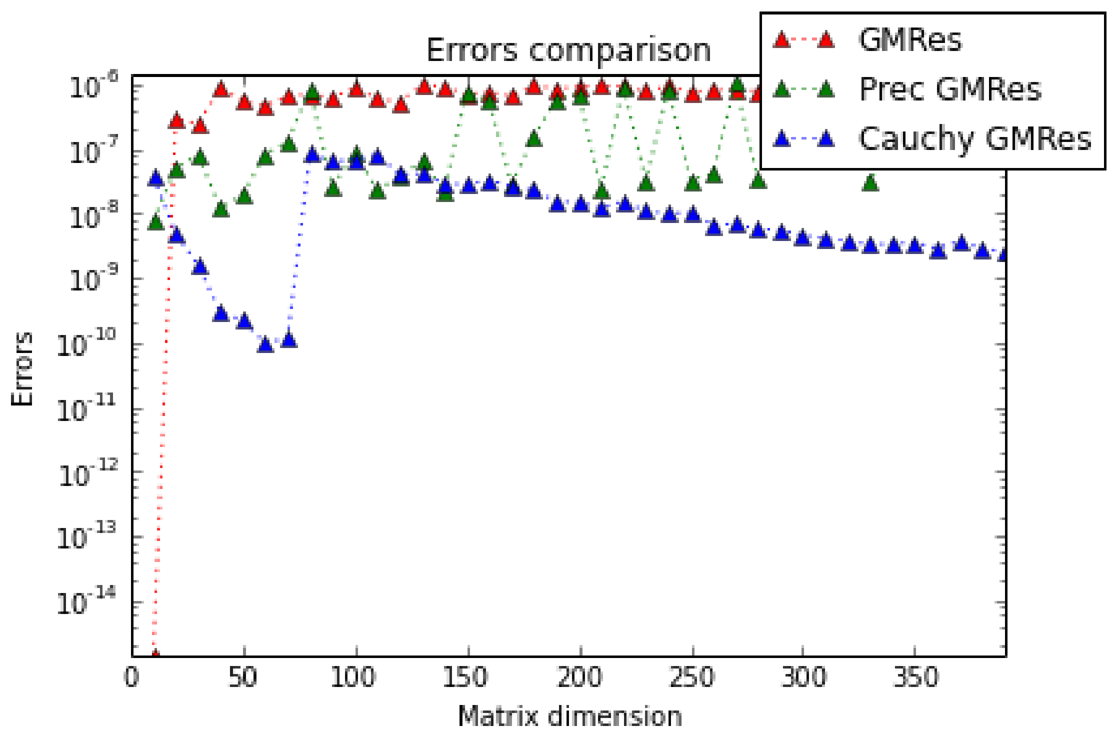
Errors comparison
GMRes
Prec GMRes
Cauchy GMRes
Errors
Matrix dimension

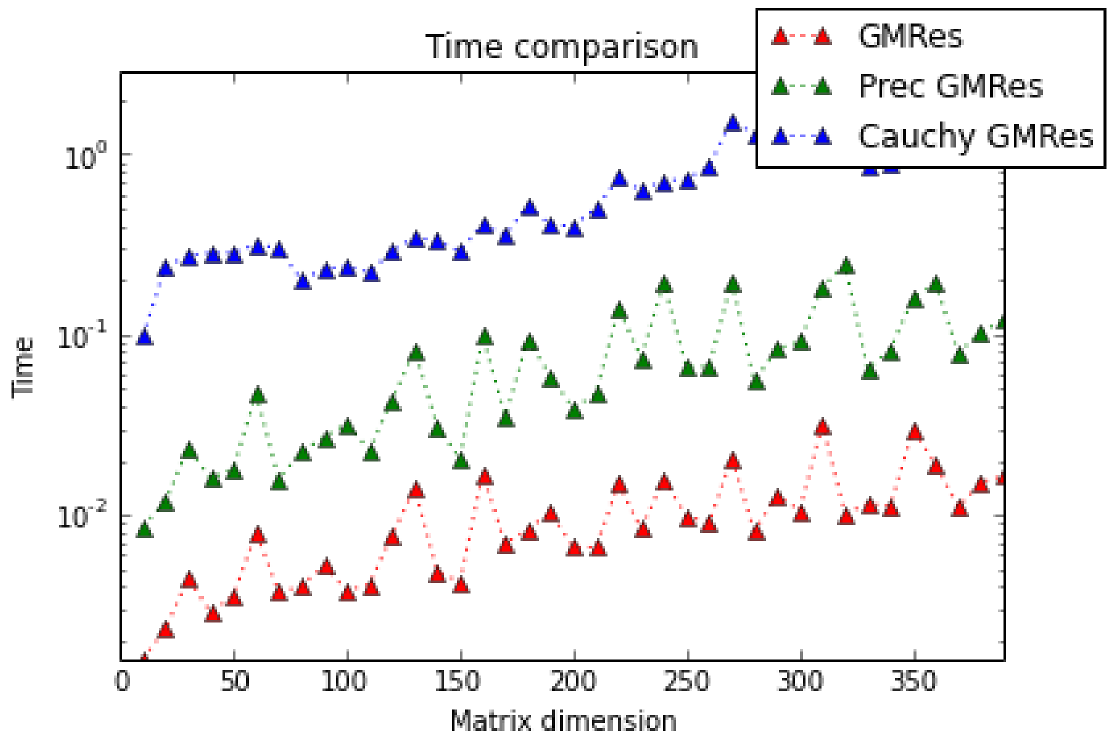

Time comparison
GMRes
Prec GMRes
Cauchy GMRes
Time
Matrix dimension

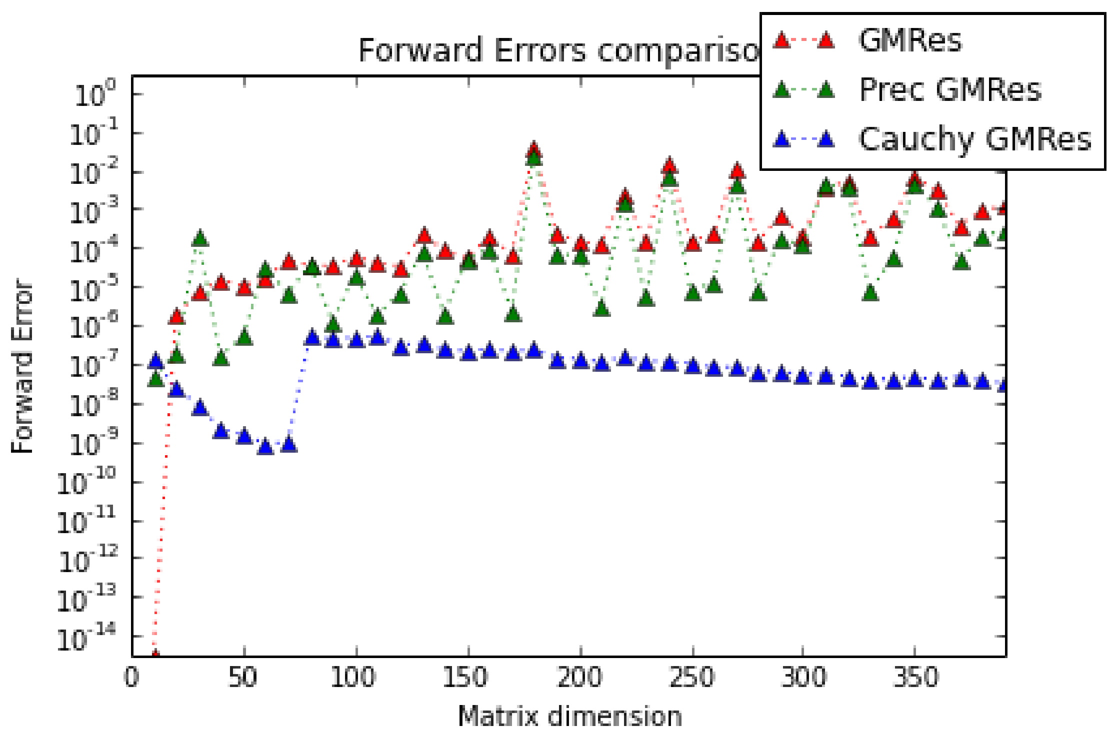

Forward Errors compariso
GMRes
Prec GMRes
Cauchy GMRes
Forward Error
Matrix dimension

N = 16 , a_tol= 1e-12

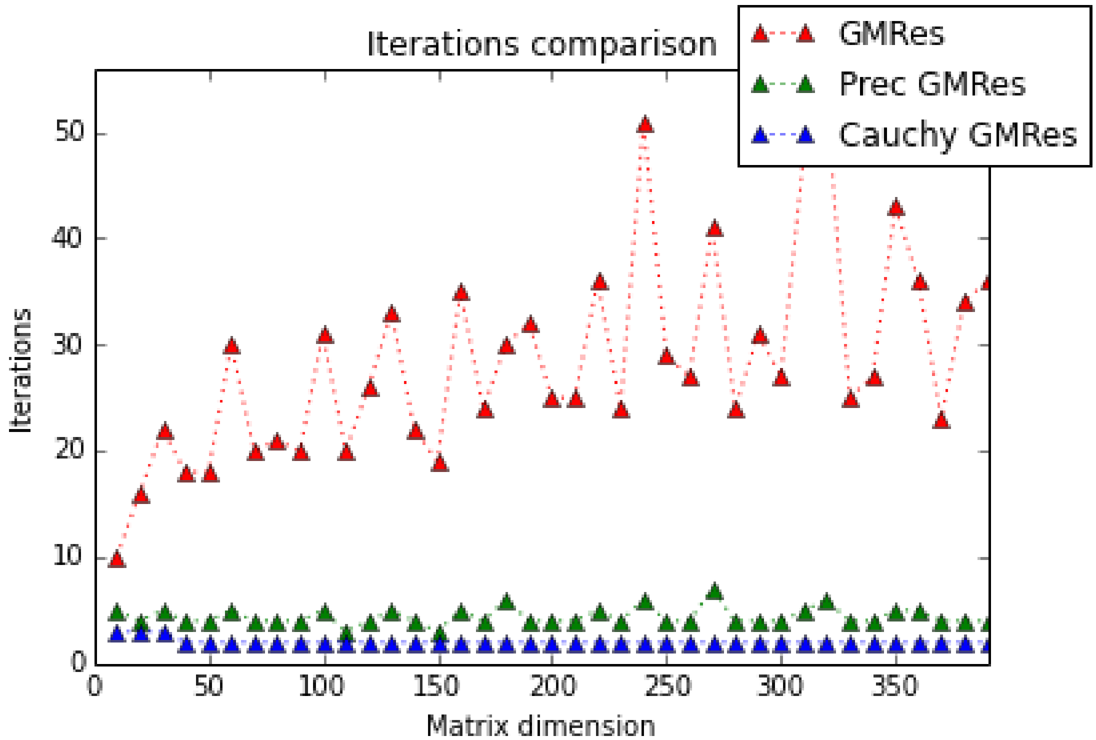


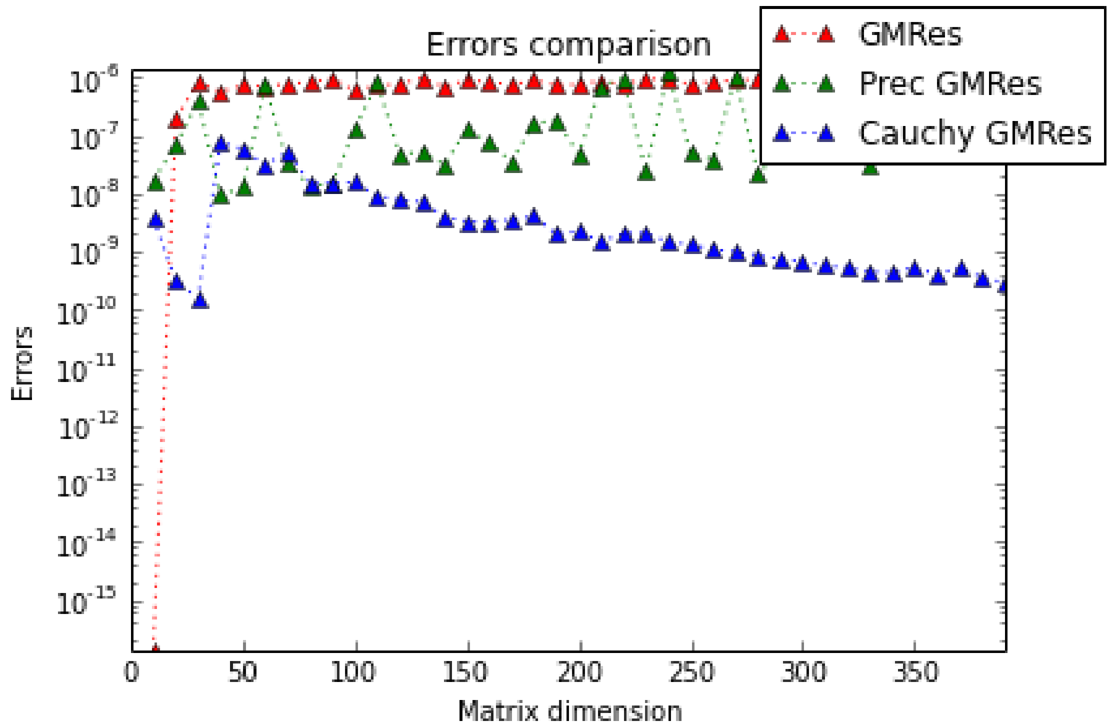


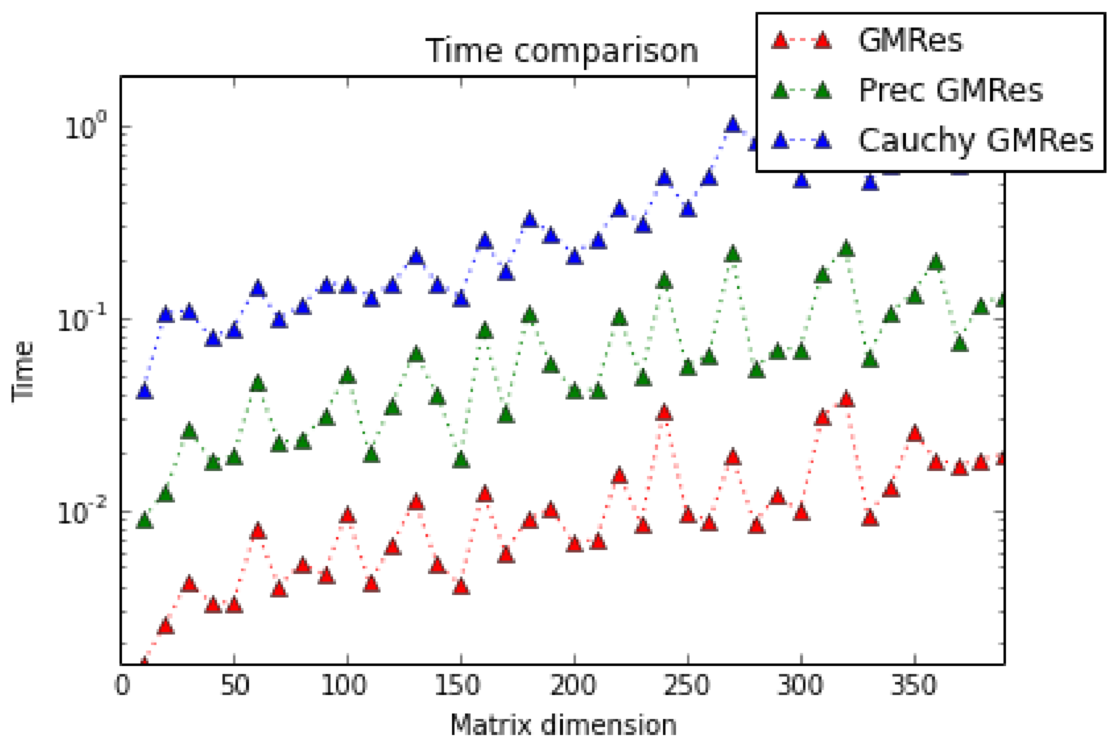

Time comparison
GMRes
Prec GMRes
Cauchy GMRes
Time
Matrix dimension

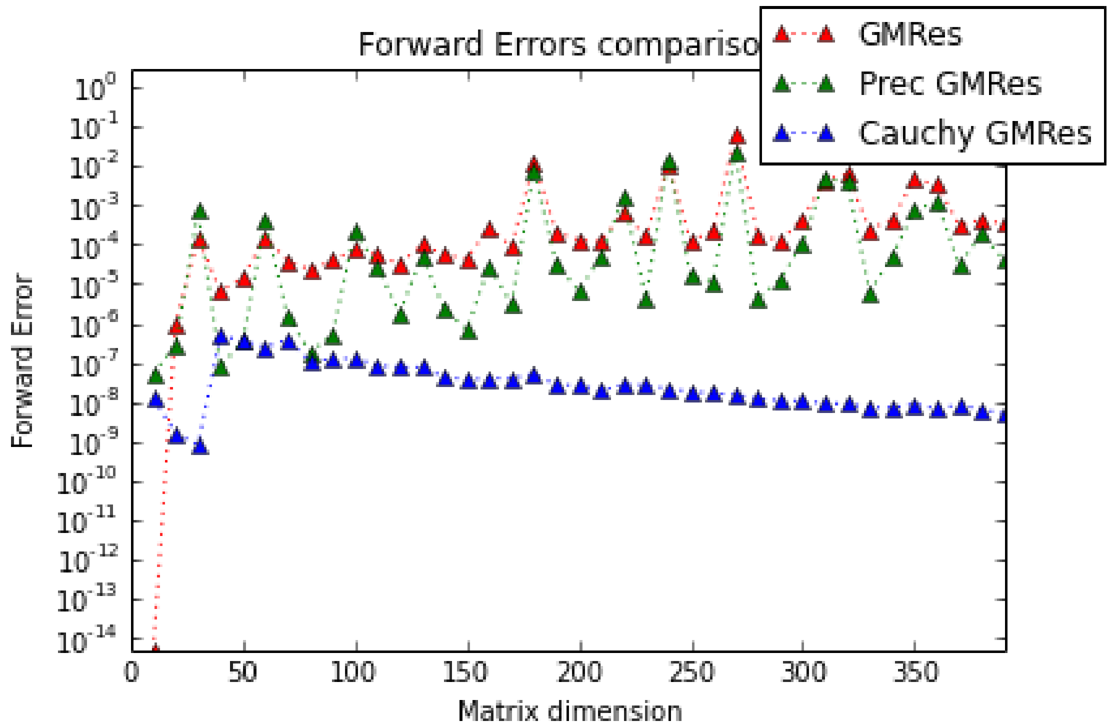

Forward Errors comparison
GMRes
Prec GMRes
Cauchy GMRes
Forward Error
Matrix dimension

N = 8 , a_tol= 1e-10

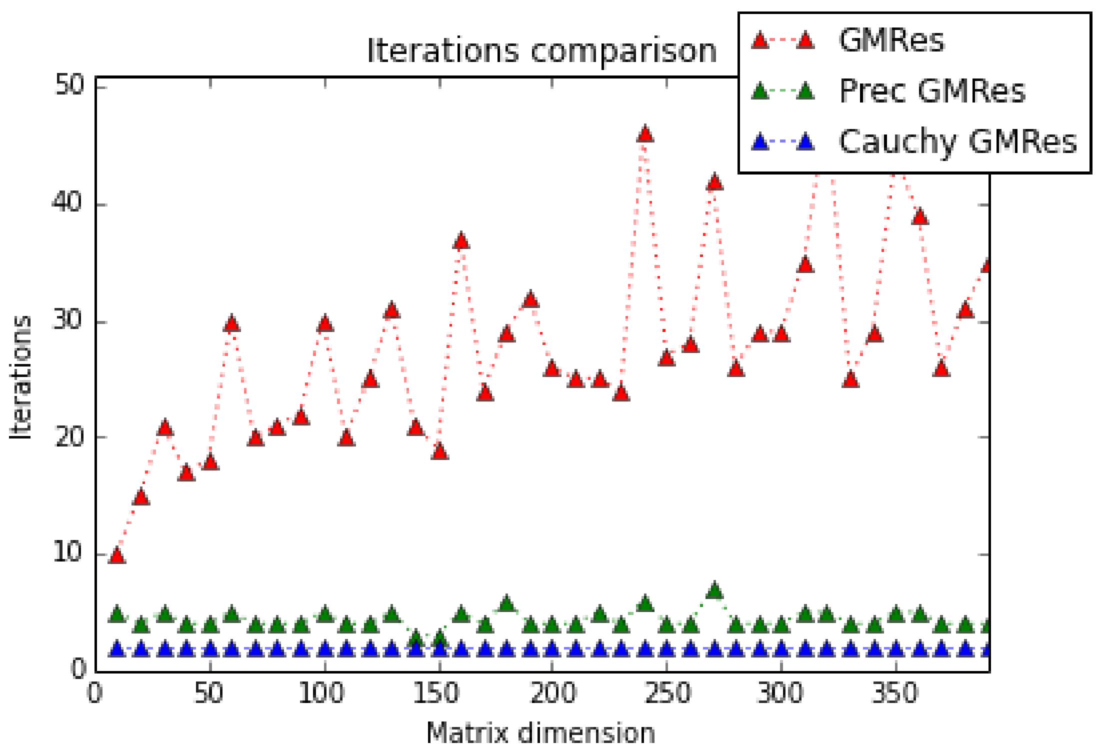

Iterations comparison
GMRes
Prec GMRes
Cauchy GMRes
Iterations
0
10
20
30
40
50
0
50
100
150
200
250
300
350
Matrix dimension

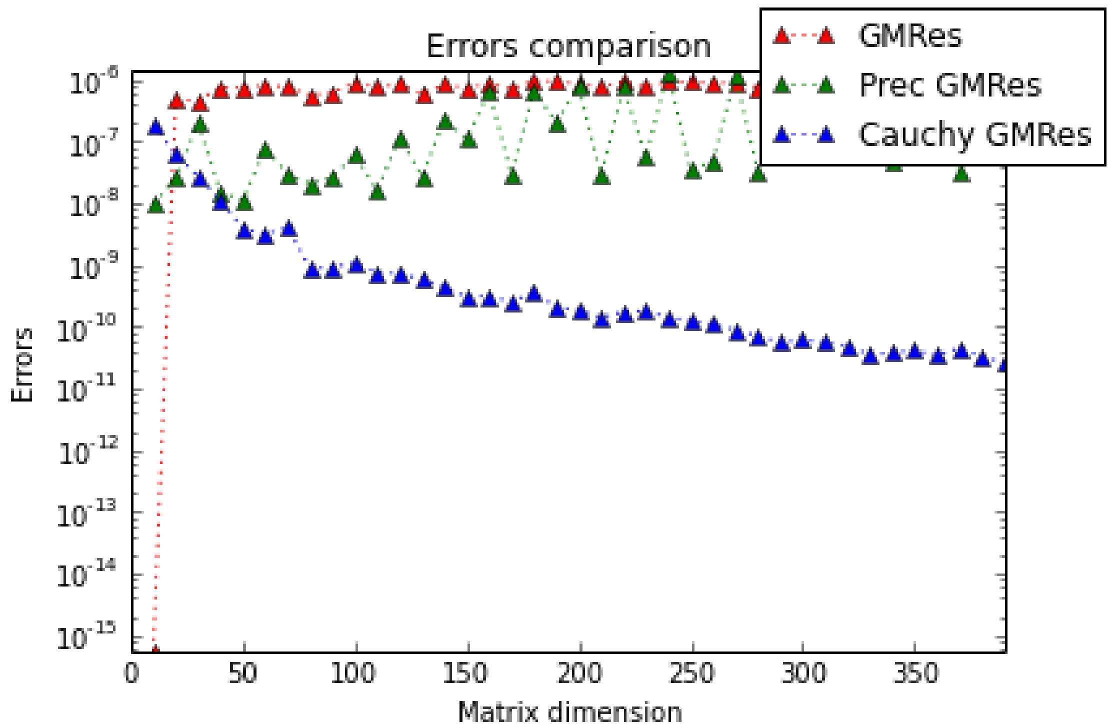

Errors comparison
GMRes
Prec GMRes
Cauchy GMRes
Errors
Matrix dimension

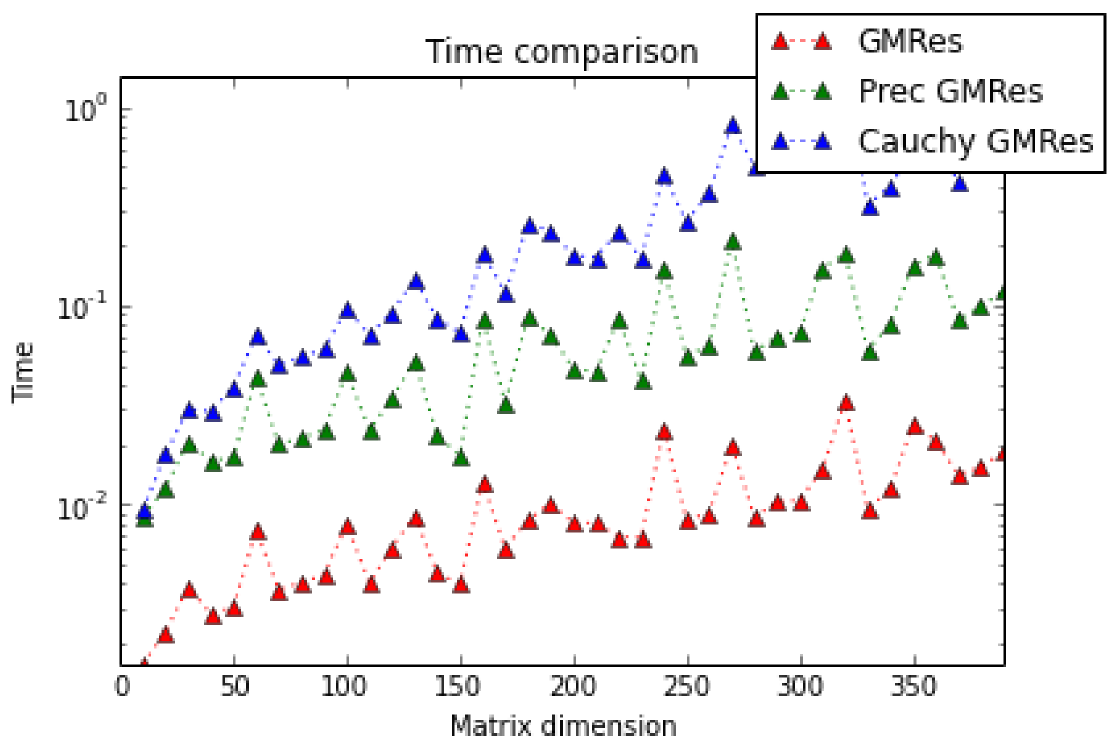

Time comparison
GMRes
Prec GMRes
Cauchy GMRes
Time
Matrix dimension

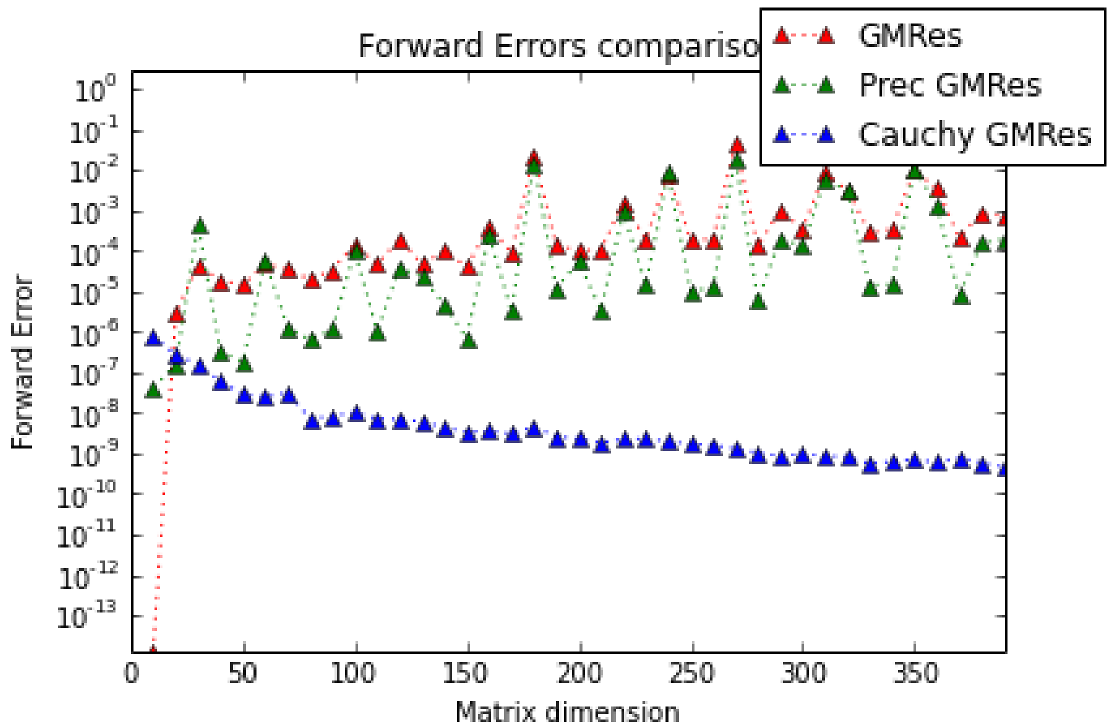

Forward Errors comparison
GMRes
Prec GMRes
Cauchy GMRes
Forward Error
Matrix dimension

N = 4 , a_tol= 1e-12

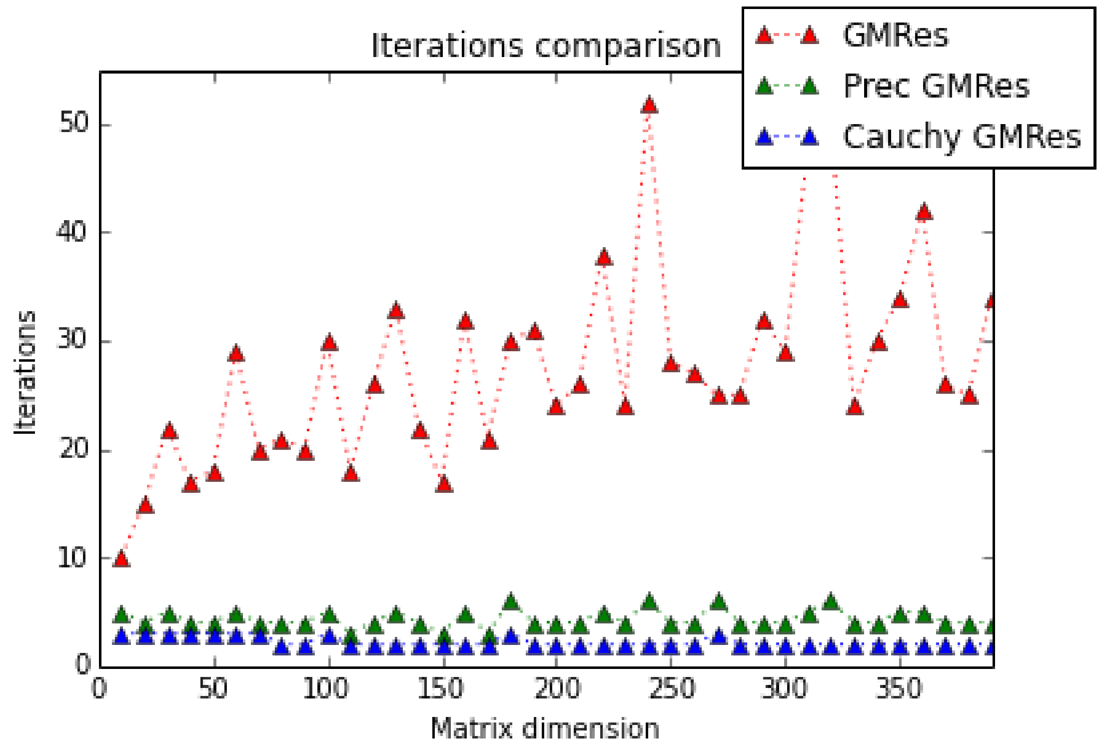

Iterations comparison
GMRes
Prec GMRes
Cauchy GMRes
Iterations
Matrix dimension

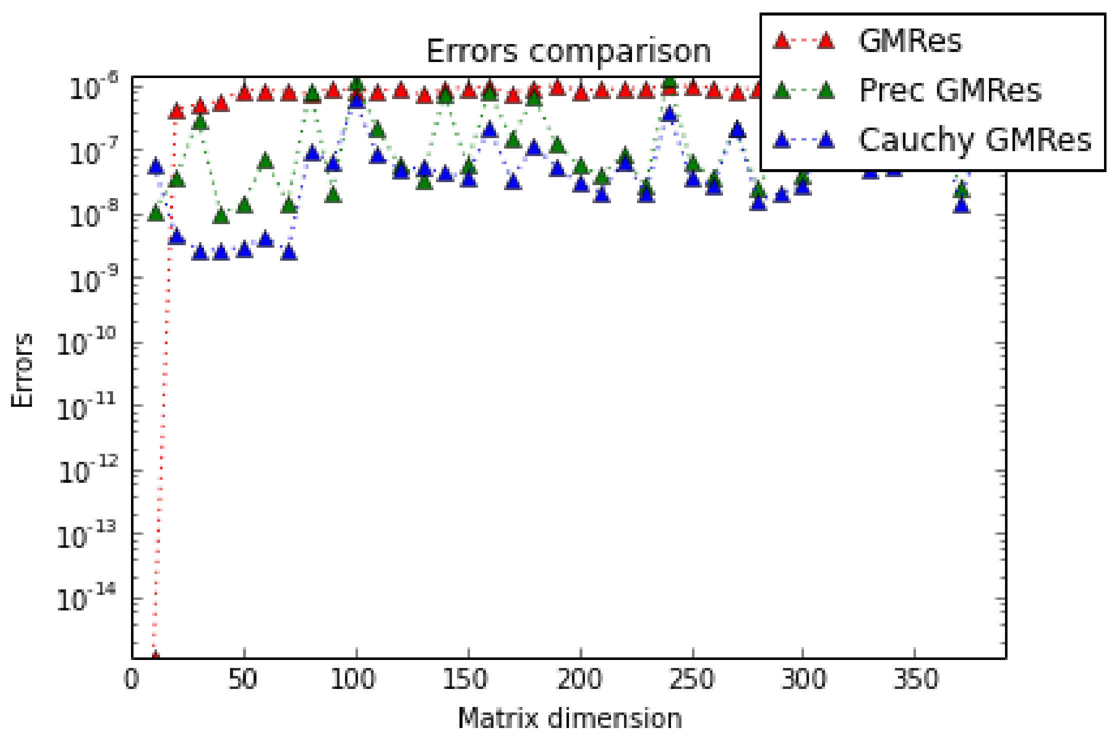

Errors comparison
GMRes
Prec GMRes
Cauchy GMRes
Errors
Matrix dimension

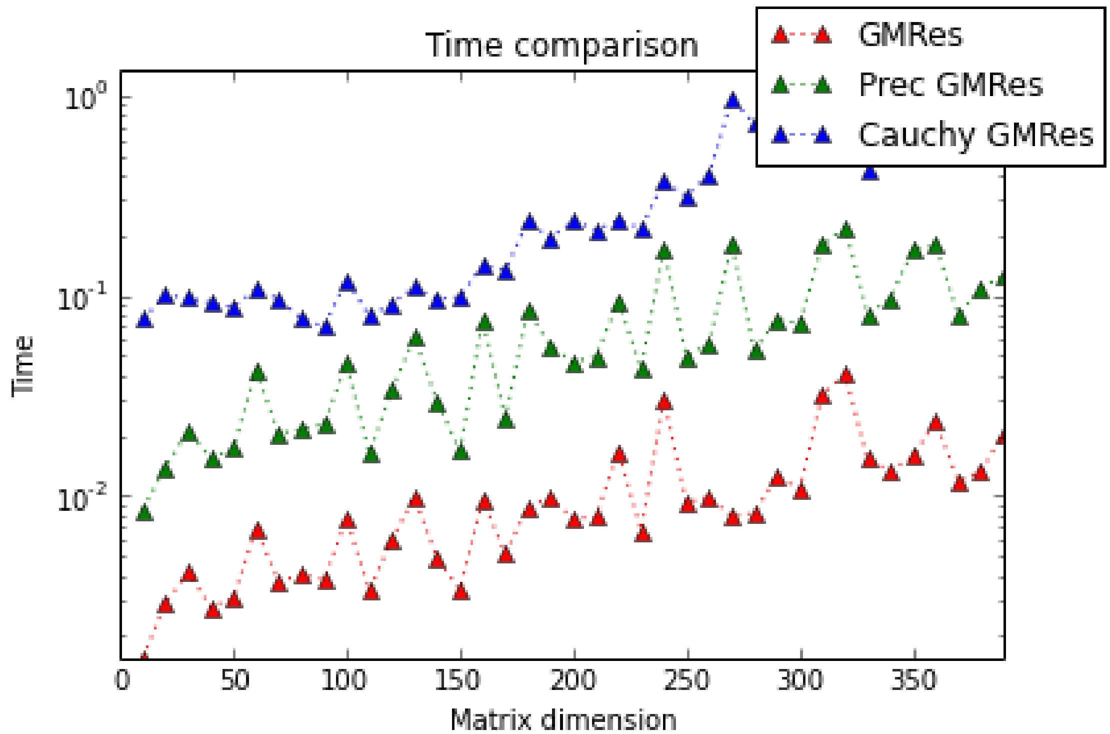

Time comparison
GMRes
Prec GMRes
Cauchy GMRes
Time
Matrix dimension

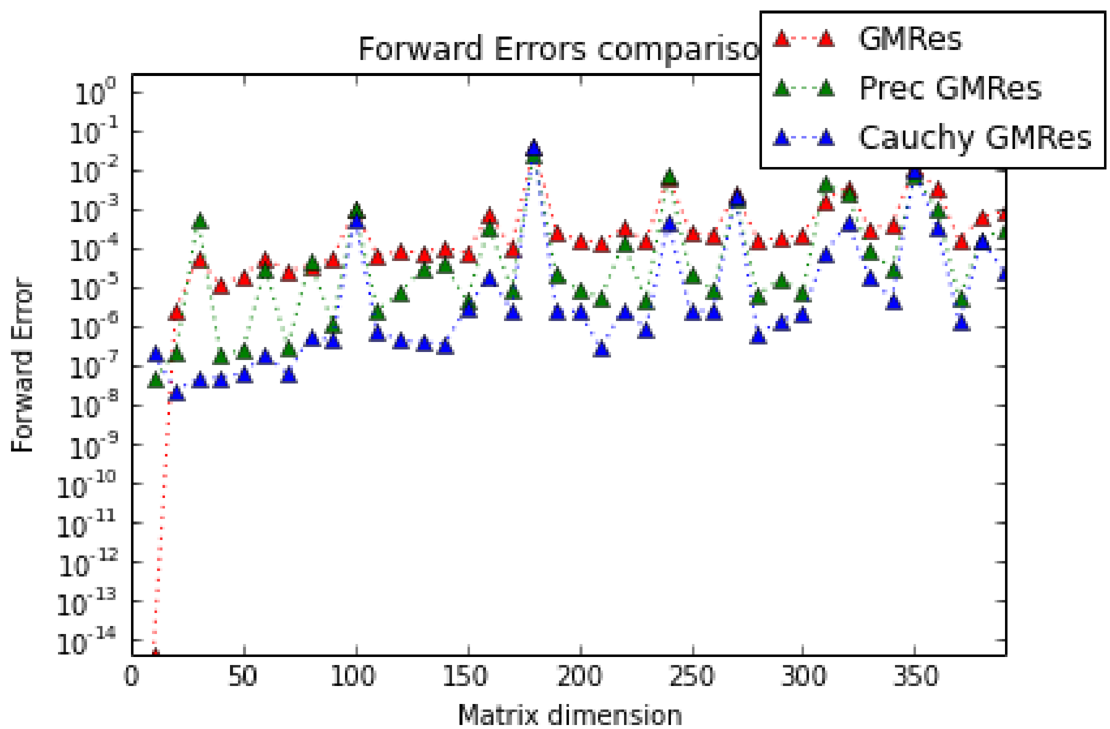

Forward Errors compariso
GMRes
Prec GMRes
Cauchy GMRes
Forward Error
Matrix dimension
0
50
100
150
200
250
300
350
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}

N = 16 , a_tol= 1e-6

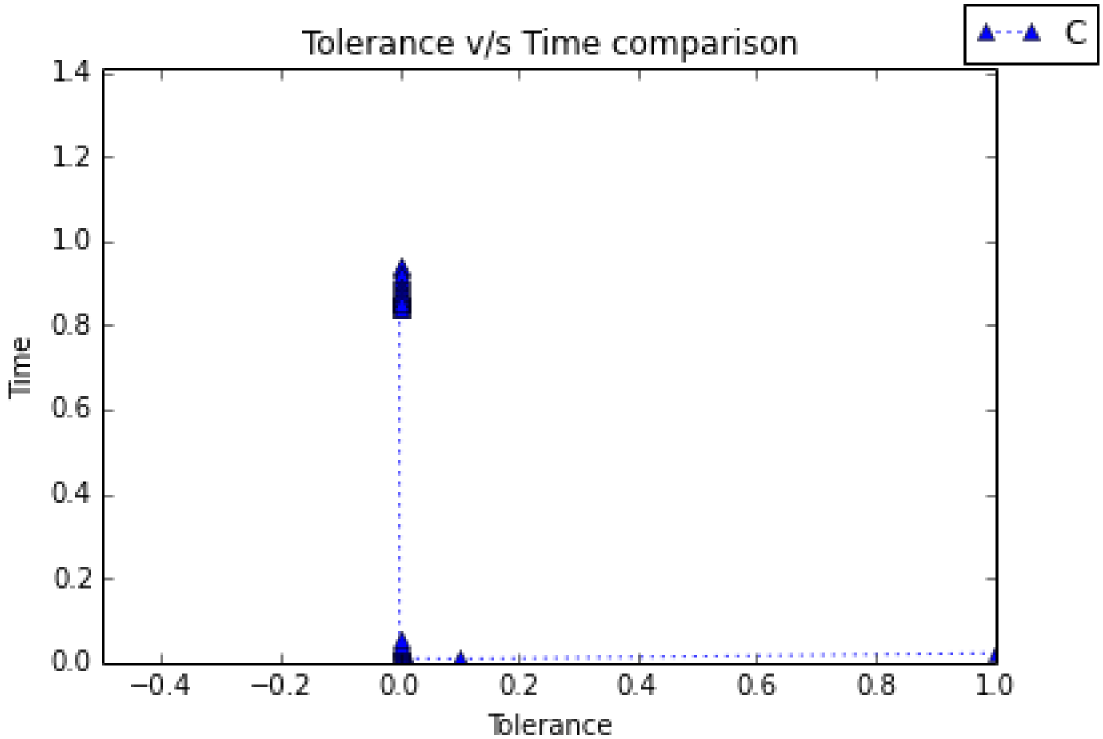

Iterations comparison
GMRes
Prec GMRes
Cauchy GMRes
Iterations
0
10
20
30
40
50
0
50
100
150
200
250
300
350
Matrix dimension


Errors comparison
GMRes
Prec GMRes
Cauchy GMRes
Errors
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
0
50
100
150
200
250
300
350
Matrix dimension


Time comparison
GMRes
Prec GMRes
Cauchy GMRes
Time
Matrix dimension


Forward Errors compariso
GMRes
Prec GMRes
Cauchy GMRes
Forward Error
Matrix dimension

```
In [52]: accuracyTime(50)
```




```
In [ ]:
```