Argonne NATIONAL LABORATORY

Forward mode source transformation AD using ADIC2

Sri Hari Krishna Narayanan and Jean Utke

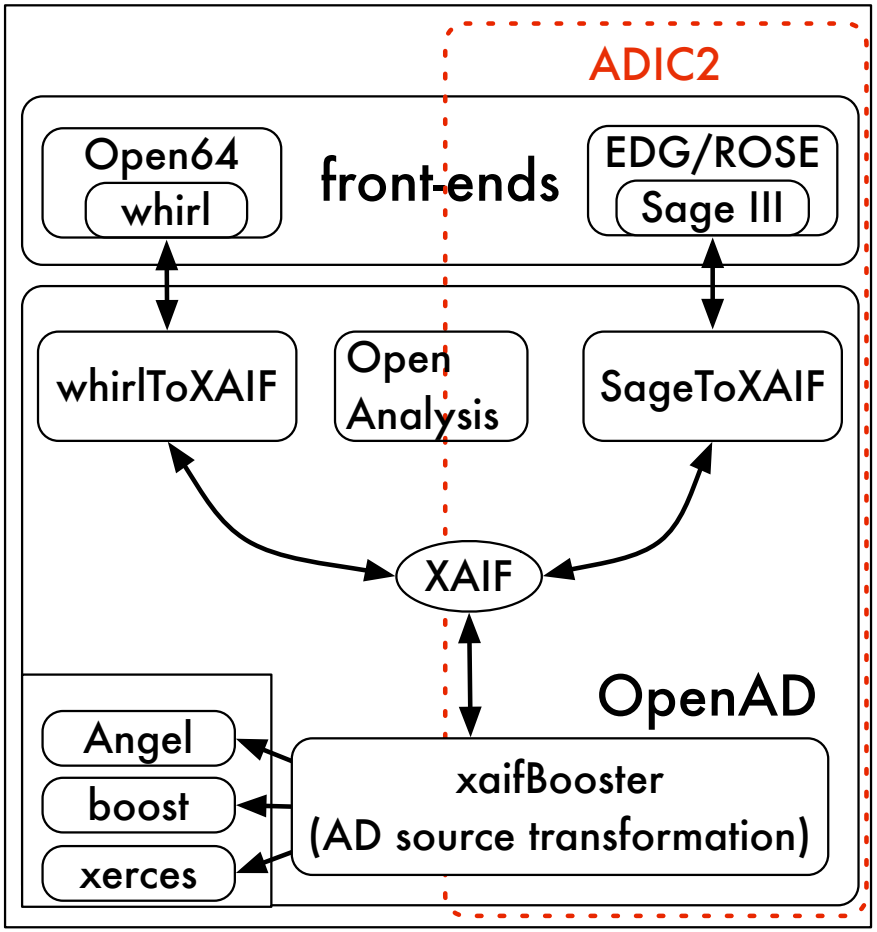
Laboratory for Advanced Numerical Simulations

Mathematics and Computer Science Division

U.S. DEPARTMENT OF ENERGY

ADIC2: Source-to-Source Automatic Differentiation Tool

- ADIC2 is an open source, source-to-source AD tool for C/C++.
- Based on the ROSE compiler framework, exploits many existing research tools.



Example of ADIC2 Generated Code : Driver

head.c

```
void mini1(double *y, double *x, int n)
{
  int i;
  for (i = 0; i < n; i=i+1) {
    y[i] = x[i] + sin(x[i]*x[i]);
  }
}
```

original_driver.c

```
void mini1(double *y, double *x,int n);

#define ARRAY_SIZE 2

int main()
{
  double x[ARRAY_SIZE], y[ARRAY_SIZE];
  int i,j;

  for (i = 0; i < ARRAY_SIZE; i++){
    x[i] = 0.5 + 0.1 * i;
  }

  // Invoke function
  mini1(y, x, ARRAY_SIZE);

  // Extract the function value
  for (i = 0; i <ARRAY_SIZE; i++) {
      printf("\n %lf " ,y[i]);
  }

  return 0;
}
```

Example of ADIC2 Generated Code:

```
1. Log into the VM.

2. cd ~/ADIC/

3. source setenv_inst.sh          -- Sets required environment variables

4. cd examples/arith_trig1/       -- Go into the example directory

5. ls                             -- What files do you see?
                                     adic_gradvec_length.h
                                     driver.c
                                     head.c
                                     Makefile
```

Example of ADIC2 Generated Code : Files

head.c

```
void mini1(double *y, double *x, int n)
{
  int i;
  for (i = 0; i < n; i=i+1) {
    y[i] = x[i] + sin(x[i]*x[i]);
  }
}
```

adic_gradvec_length.h

```
#define ADIC_GRADVEC_LENGTH 2
```

Makefile

1. Invokes ADIC to generate derivative code
2. Compiles original code, derivative code, driver and runtime library.

driver.c

```
#define ARRAY_SIZE ADIC_GRADVEC_LENGTH

void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x,int n);
int main()
{
  double x[ARRAY_SIZE], y[ARRAY_SIZE];
  DERIV_TYPE ad_x[ARRAY_SIZE], ad_y[ARRAY_SIZE];
  int i,j;
  for (i = 0; i < ARRAY_SIZE; i++){
    x[i] = 0.5 + 0.1 * i;
  }

  // Set independent variables
  ADIC_SetForwardMode();
  ADIC_SetDepArray(ad_y, ARRAY_SIZE);
  ADIC_SetIndepArray(ad_x, ARRAY_SIZE);
  ADIC_SetIndepDone();

  // Initialize the value of the independent variable ad_x
  for (i = 0; i < ARRAY_SIZE; i++){
    DERIV_val(ad_x[i]) =  x[i];
  }

  // Invoke AD function
  ad_mini1(ad_y, ad_x, ARRAY_SIZE);

  // Extract the Jacobian
  for (i = 0; i <ARRAY_SIZE; i++) {
    printf("\n[");
    for (j = 0; j <ADIC_GRADVEC_LENGTH; j++) {
      printf(" %lf " ,DERIV_grad(ad_y[i])[j]);
    }
    printf("]");
  }
  ADIC_Finalize();
  return 0;
}
```

Example of ADIC2 Generated Code:

```
1. Log into the VM.

2. cd ~/ADIC/

3. source setenv_inst.sh          -- Sets required environment variables

4. cd examples/arith_trig1/       -- Go into the example directory

5. ls                             -- What files do you see?

6. make                           -- Generate output and compile it

7. ls *.c *.h                     -- What are the new files?
                                     head.cn.xb.pp.c
                                     ad_grad_saxpy-n_dense.h
                                     runtime_dense/ad_grad.h
                                     runtime_dense/ad_types.h
                                     runtime_dense/adic_gradvec_length.h
```

Example of ADIC2 Generated Code

head.cn.xb.pp.c

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y, DERIV_TYPE *x, int n)
{
  int ad_i;
  double ad_acc_1, ad_acc_0, ad_lin_2;
  double ad_lin_1, ad_lin_0, ad_aux_0;
  for (ad_i = 0, ad_i = 0; ad_i < n; ad_i = ad_i + 1) {
    DERIV_TYPE ad_prp_2;
    ADIC_Initialize(&ad_prp_2);
    DERIV_TYPE ad_prp_1;
    ADIC_Initialize(&ad_prp_1);
    DERIV_TYPE ad_prp_0;
    ADIC_Initialize(&ad_prp_0);
    ad_aux_0 = DERIV_val(x[ad_i]) * DERIV_val(x[ad_i]);
    ad_lin_0 = DERIV_val(x[ad_i]);
    ad_lin_1 = DERIV_val(x[ad_i]);
    ad_lin_2 = cos(ad_aux_0);
    DERIV_val(y[ad_i]) = DERIV_val(x[ad_i]) + sin(ad_aux_0);
    ad_acc_0 = ad_lin_0 * ad_lin_2;
    ad_acc_1 = ad_lin_1 * ad_lin_2;
    ADIC_SetDeriv(x[ad_i],ad_prp_0);
    ADIC_SetDeriv(x[ad_i],ad_prp_1);
    ADIC_SetDeriv(x[ad_i],ad_prp_2);
    ADIC_Sax_3(1,ad_prp_0,ad_acc_0,ad_prp_1,ad_acc_1,ad_prp_2,y[ad_i]);
  }
}
```

Example of ADIC2 Generated Code

head.cn.xb.pp.c

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y, DERIV_TYPE *x, int n)
{
  int ad_i;
  double ad_acc_1, ad_acc_0, ad_lin_2;
  double ad_lin_1, ad_lin_0, ad_aux_0;
  for (ad_i = 0, ad_i = 0; ad_i < n; ad_i = ad_i + 1) {
    DERIV_TYPE ad_prp_2;
    ADIC_Initialize(&ad_prp_2);
    DERIV_TYPE ad_prp_1;
    ADIC_Initialize(&ad_prp_1);
    DERIV_TYPE ad_prp_0;
    ADIC_Initialize(&ad_prp_0);
    ad_aux_0 = DERIV_val(x[ad_i]) * DERIV_val(x[ad_i]);
    ad_lin_0 = DERIV_val(x[ad_i]);
    ad_lin_1 = DERIV_val(x[ad_i]);
    ad_lin_2 = cos(ad_aux_0);
    DERIV_val(y[ad_i]) = DERIV_val(x[ad_i]) + sin(ad_aux_0);
    ad_acc_0 = ad_lin_0 * ad_lin_2;
    ad_acc_1 = ad_lin_1 * ad_lin_2;
    ADIC_SetDeriv(x[ad_i],ad_prp_0);
    ADIC_SetDeriv(x[ad_i],ad_prp_1);
    ADIC_SetDeriv(x[ad_i],ad_prp_2);
    ADIC_Sax_3(1,ad_prp_0,ad_acc_0,ad_prp_1,ad_acc_1,ad_prp_2,y[ad_i]);
  }
}
```

ad_types.h

```
typedef struct {
  double val;
  double grad[ADIC_GRADVEC_LENGTH];
} DERIV_TYPE;
```

```
#define DERIV_val(a)      (a).val
#define DERIV_grad(a)     (a).grad
```

adic_gradvec_length.h

```
#define ADIC_GRADVEC_LENGTH 2
```

Example of ADIC2 Generated Code

head.cn.xb.pp.c

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y, DERIV_TYPE *x, int n)
{
  int ad_i;
  double ad_acc_1, ad_acc_0, ad_lin_2;
  double ad_lin_1, ad_lin_0, ad_aux_0;
  for (ad_i = 0, ad_i = 0; ad_i < n; ad_i = ad_i + 1) {
    DERIV_TYPE ad_prp_2;
    ADIC_Initialize(&ad_prp_2);
    DERIV_TYPE ad_prp_1;
    ADIC_Initialize(&ad_prp_1);
    DERIV_TYPE ad_prp_0;
    ADIC_Initialize(&ad_prp_0);
    ad_aux_0 = DERIV_val(x[ad_i]) * DERIV_val(x[ad_i]);
    ad_lin_0 = DERIV_val(x[ad_i]);
    ad_lin_1 = DERIV_val(x[ad_i]);
    ad_lin_2 = cos(ad_aux_0);
    DERIV_val(y[ad_i]) = DERIV_val(x[ad_i]) + sin(ad_aux_0);
    ad_acc_0 = ad_lin_0 * ad_lin_2;
    ad_acc_1 = ad_lin_1 * ad_lin_2;
    ADIC_SetDeriv(x[ad_i],ad_prp_0);
    ADIC_SetDeriv(x[ad_i],ad_prp_1);
    ADIC_SetDeriv(x[ad_i],ad_prp_2);
    ADIC_Sax_3(1,ad_prp_0,ad_acc_0,ad_prp_1,ad_acc_1,ad_prp_2,y[ad_i]);
  }
}
```

ad_grad.h

```
void ADIC_SetDeriv(DERIV_TYPE &src,
                   DERIV_TYPE &tgt) {
  int i;
  double *grad_t = DERIV_grad(tgt);
  double *grad_s = DERIV_grad(src);
  for(i = 0; i < ADIC_GRADVEC_LENGTH; i++) {
    grad_t[i] = grad_s[i];
  }
}
```

Example of ADIC2 Generated Code

head.cn.xb.pp.c

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y, DERIV_TYPE *x, int n)
{
  int ad_i;
  double ad_acc_1, ad_acc_0, ad_lin_2;
  double ad_lin_1, ad_lin_0, ad_aux_0;
  for (ad_i = 0, ad_i = 0; ad_i < n; ad_i = ad_i + 1) {
    DERIV_TYPE ad_prp_2;
    ADIC_Initialize(&ad_prp_2);
    DERIV_TYPE ad_prp_1;
    ADIC_Initialize(&ad_prp_1);
    DERIV_TYPE ad_prp_0;
    ADIC_Initialize(&ad_prp_0);
    ad_aux_0 = DERIV_val(x[ad_i]) * DERIV_val(x[ad_
    ad_lin_0 = DERIV_val(x[ad_i]);
    ad_lin_1 = DERIV_val(x[ad_i]);
    ad_lin_2 = cos(ad_aux_0);
    DERIV_val(y[ad_i]) = DERIV_val(x[ad_i]) + sin(a
    ad_acc_0 = ad_lin_0 * ad_lin_2;
    ad_acc_1 = ad_lin_1 * ad_lin_2;
    ADIC_SetDeriv(x[ad_i],ad_prp_0);
    ADIC_SetDeriv(x[ad_i],ad_prp_1);
    ADIC_SetDeriv(x[ad_i],ad_prp_2);
    ADIC_Sax_3(1,ad_prp_0,ad_acc_0,ad_prp_1,ad_acc_1,ad_prp_2,y[ad_i]);
  }
}
```

ad_grad_saxpy-n_dense.h

```
void ADIC_Sax_3(  double a1, DERIV_TYPE &x1,
                  double a2, DERIV_TYPE &x2,
                  double a3, DERIV_TYPE &x3, DERIV_TYPE &tgt)
{
  int i;
  double *grad1 = DERIV_grad(x1), *grad2 = DERIV_grad(x2),
         *grad3 = DERIV_grad(x3), *gradt = DERIV_grad(tgt);
  for (i = 0; i < ADIC_GRADVEC_LENGTH; i++) {
    gradt[i] = a1 * grad1[i] + a2 * grad2[i] + a3 * grad3[i];
  }
}
```

Example of ADIC2 Generated Code

- The grad field of y forms the Jacobian Matrix.
- The driver initializes x and y appropriately
- The derivative code computes y[i].grad

$$\begin{bmatrix} \frac{\partial y_0}{\partial x_0} & \frac{\partial y_0}{\partial x_1} \\ \frac{\partial y_1}{\partial x_0} & \frac{\partial y_1}{\partial x_1} \end{bmatrix} = \begin{bmatrix} (y[0].grad)[0] & (y[0].grad)[1] \\ (y[1].grad)[0] & (y[1].grad)[1] \end{bmatrix}$$

Example of ADIC2 Generated Code

```
1. Log into the VM.

2. cd ~/ADIC/

3. source setenv_inst.sh          -- Sets required environment variables

4. cd examples/arith_trig1/       -- Go into the example directory

5. ls                             -- What files do you see?

6. make                           -- Generate output and compile it

7. ls *.c *.h                     -- What are the new files?

8. ./driver                       -- How was the output generated?
```

Example of ADIC2 Generated Code : Driver

```
#define ARRAY_SIZE ADIC_GRADVEC_LENGTH
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x,int n);
int main()
{
  double x[ARRAY_SIZE], y[ARRAY_SIZE];
  DERIV_TYPE ad_x[ARRAY_SIZE], ad_y[ARRAY_SIZE];
  int i,j;
  for (i = 0; i < ARRAY_SIZE; i++){
    x[i] = 0.5 + 0.1 * i;
  }

  // Set independent variables
  ADIC_SetForwardMode();
  ADIC_SetDepArray(ad_y, ARRAY_SIZE);
  ADIC_SetIndepArray(ad_x, ARRAY_SIZE);
  ADIC_SetIndepDone();

  // Initialize the value of the independent variable ad_x
  for (i = 0; i < ARRAY_SIZE; i++){
    DERIV_val(ad_x[i]) =  x[i];
  }

  // Invoke AD function
  ad_mini1(ad_y, ad_x, ARRAY_SIZE);

  // Extract the Jacobian
  for (i = 0; i <ARRAY_SIZE; i++) {
    printf("\n[");
    for (j = 0; j <ADIC_GRADVEC_LENGTH; j++) {
      printf(" %lf " ,DERIV_grad(ad_y[i])[j]);
    }
    printf("]");
  }
  ADIC_Finalize();
  return 0;
}
```

y[0].grad	0.0	0.0
y[1].grad	0.0	0.0

x[0].grad	1.0	0.0
x[1].grad	0.0	1.0

x[0].val	0.5
x[1].val	0.6

y[0].val	0.0
y[1].val	0.0

What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```

x[0].val	0.5
x[1].val	0.6

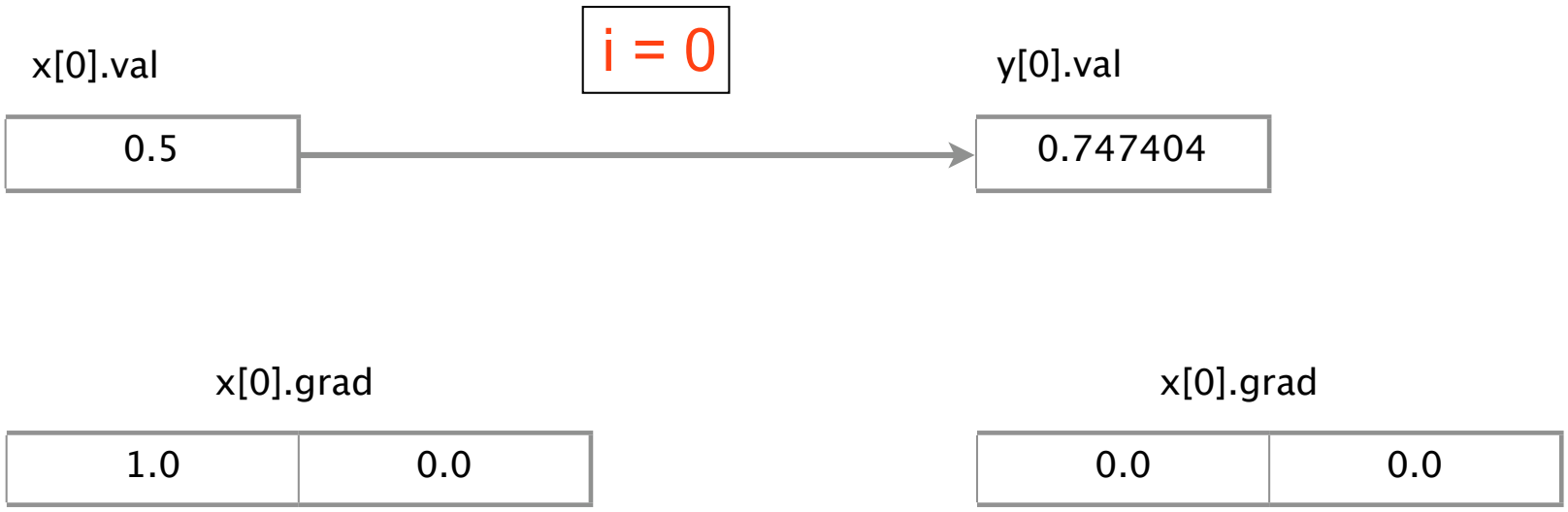
y[0].val	0.0
y[1].val	0.0

x[0].grad	1.0	0.0
x[1].grad	0.0	1.0

y[0].grad	0.0	0.0
y[1].grad	0.0	0.0

What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```

i = 0

x[0].val

0.5

y[0].val

0.747404

x[0].grad

1.0	0.0

y[0].grad

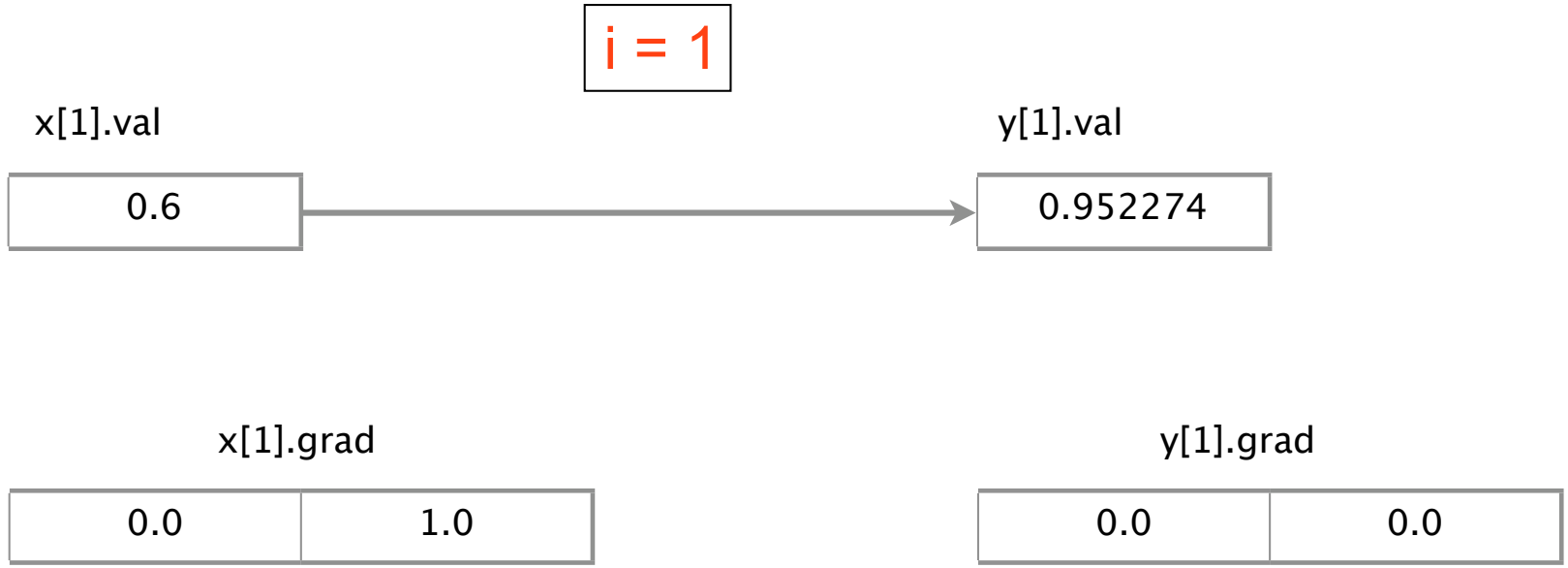
1.968912	0.0

1 * 1.0 + 0.484456 * 1.0 + 0.484456 *1.0

1 * 0.0 + 0.484456 * 0.0 + 0.484456 *0.0

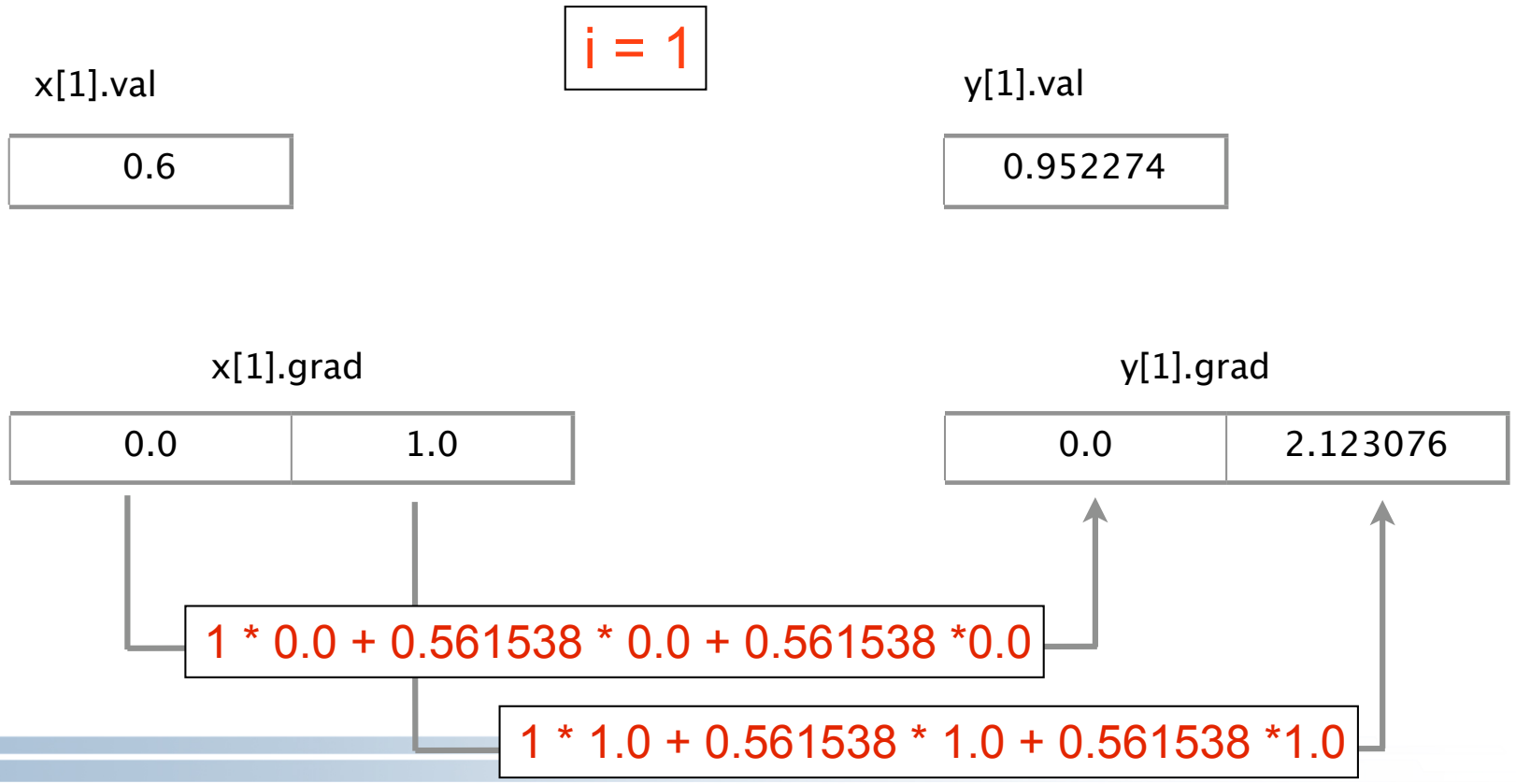
What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```

y[0].val	0.747404
y[1].val	0.952274

y[0].grad	1.968912	0.0
y[1].grad	0.0	2.123076

$$\begin{bmatrix} \frac{\partial y_0}{\partial x_0} & \frac{\partial y_0}{\partial x_1} \\ \frac{\partial y_1}{\partial x_0} & \frac{\partial y_1}{\partial x_1} \end{bmatrix} = \begin{bmatrix} (y[0].grad)[0] & (y[0].grad)[1] \\ (y[1].grad)[0] & (y[1].grad)[1] \end{bmatrix}$$

Conclusions

- ADIC2
  - https://trac.mcs.anl.gov/projects/ADIC
- Automatic Differentiation
  - http://www.autodiff.org/
  - www.mcs.anl.gov/autodiff/

Argonne NATIONAL LABORATORY

Exploiting Sparse Jacobians with ADIC2

Sri Hari Krishna Narayanan and Jean Utke

Laboratory for Advanced Numerical Simulations

Mathematics and Computer Science Division

U.S. DEPARTMENT OF ENERGY

Recall: Example of ADIC2 Generated Code : Driver

```
#define ARRAY_SIZE ADIC_GRADVEC_LENGTH
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x,int n);
int main()
{
  double x[ARRAY_SIZE], y[ARRAY_SIZE];
  DERIV_TYPE ad_x[ARRAY_SIZE], ad_y[ARRAY_SIZE];
  int i,j;
  for (i = 0; i < ARRAY_SIZE; i++){
    x[i] = 0.5 + 0.1 * i;
  }

  // Set independent variables
  ADIC_SetForwardMode();
  ADIC_SetDepArray(ad_y, ARRAY_SIZE);
  ADIC_SetIndepArray(ad_x, ARRAY_SIZE);
  ADIC_SetIndepDone();

  // Initialize the value of the independent variable ad_x
  for (i = 0; i < ARRAY_SIZE; i++){
    DERIV_val(ad_x[i]) =  x[i];
  }

  // Invoke AD function
  ad_mini1(ad_y, ad_x, ARRAY_SIZE);

  // Extract the Jacobian
  for (i = 0; i <ARRAY_SIZE; i++) {
    printf("\n[");
    for (j = 0; j <ADIC_GRADVEC_LENGTH; j++) {
      printf(" %lf " ,DERIV_grad(ad_y[i])[j]);
    }
    printf("]");
  }
  ADIC_Finalize();
  return 0;
}
```

y[0].grad	0.0	0.0
y[1].grad	0.0	0.0

x[0].grad	1.0	0.0
x[1].grad	0.0	1.0

x[0].val	0.5
x[1].val	0.6

y[0].val	0.0
y[1].val	0.0

Recall: What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```

x[0].val	0.5
x[1].val	0.6

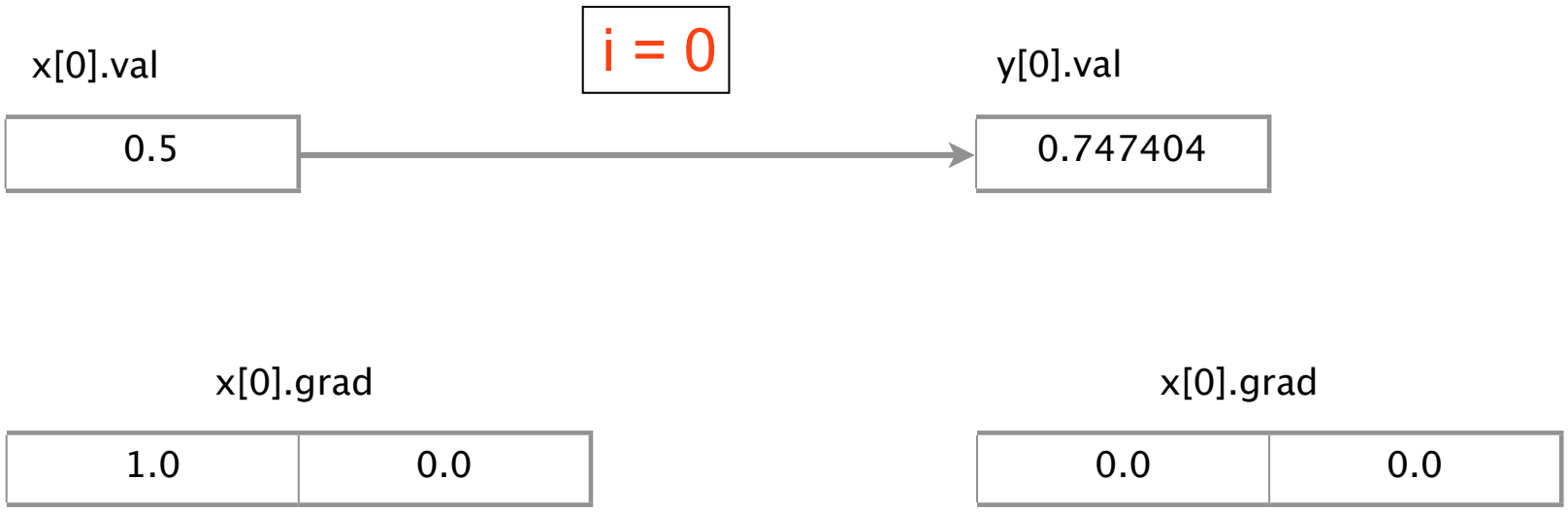
y[0].val	0.0
y[1].val	0.0

x[0].grad	1.0	0.0
x[1].grad	0.0	1.0

y[0].grad	0.0	0.0
y[1].grad	0.0	0.0

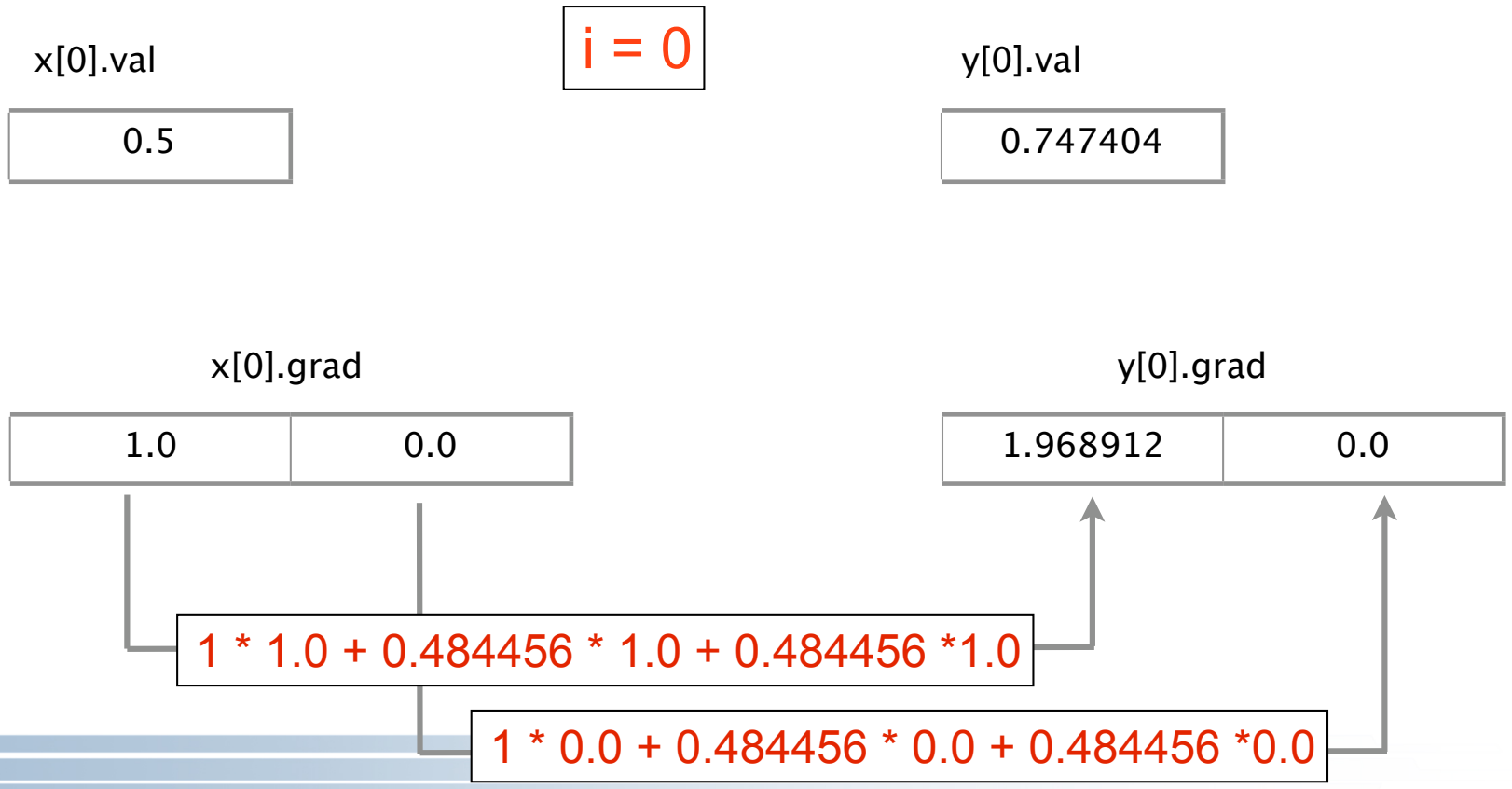
Recall: What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



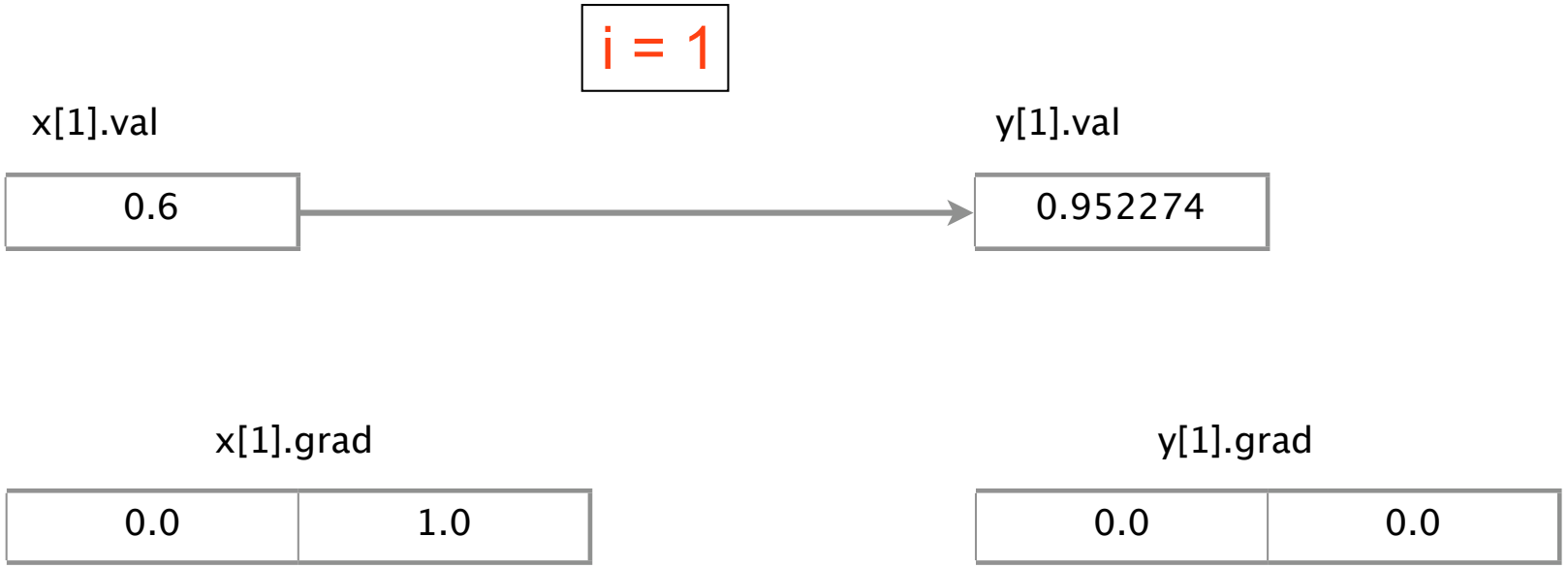
Recall: What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



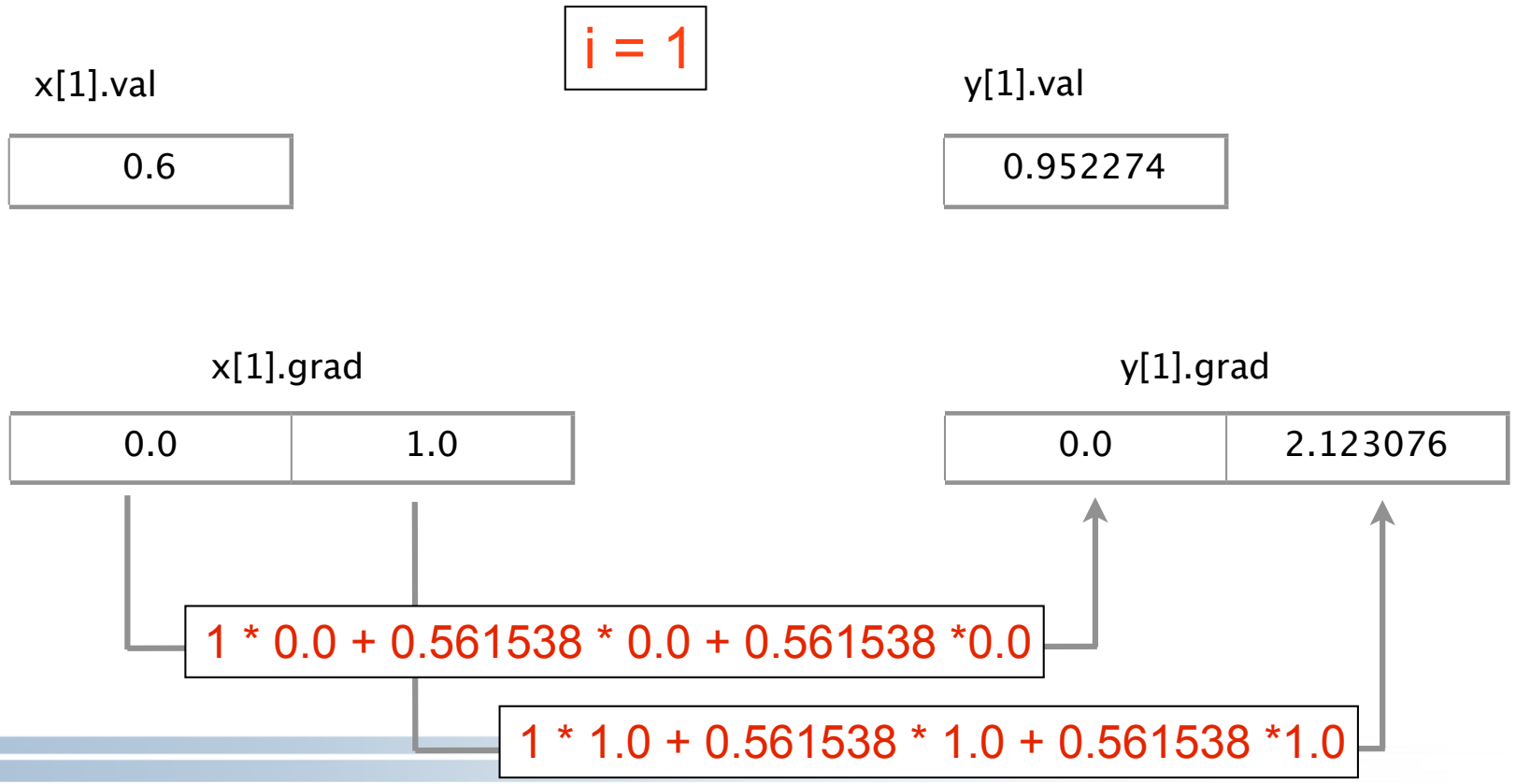
Recall: What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



Recall: What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



Recall: What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```

y[0].val	0.747404
y[1].val	0.952274

y[0].grad	1.968912	0.0
y[1].grad	0.0	2.123076

$$\begin{bmatrix} \frac{\partial y_0}{\partial x_0} & \frac{\partial y_0}{\partial x_1} \\ \frac{\partial y_1}{\partial x_0} & \frac{\partial y_1}{\partial x_1} \end{bmatrix} = \begin{bmatrix} (y[0].grad)[0] & (y[0].grad)[1] \\ (y[1].grad)[0] & (y[1].grad)[1] \end{bmatrix}$$

The problem with computation of dense Jacobians

- All elements of the Jacobian are accessed even if the Jacobian is sparse!
- In this example, only the diagonal elements are non-zero.

$$\begin{bmatrix} \frac{\partial y_0}{\partial x_0} & \frac{\partial y_0}{\partial x_1} & \cdots & \frac{\partial y_0}{\partial x_{n-1}} \\ \frac{\partial y_1}{\partial x_0} & \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{n-1}}{\partial x_0} & \frac{\partial y_{n-1}}{\partial x_1} & \cdots & \frac{\partial y_{n-1}}{\partial x_{n-1}} \end{bmatrix} = \begin{bmatrix} (y[0].grad)[0] & 0 & 0 & \cdots & 0 \\ 0 & (y[1].grad)[1] & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & & \ddots & 0 \\ 0 & 0 & \cdots & 0 & (y[n-1].grad)[n-1] \end{bmatrix}$$

- The solution is to compress the matrix using graph partitioning/coloring techniques in ColPack.

Partitioning Of The Jacobian Into Structurally Orthogonal Columns

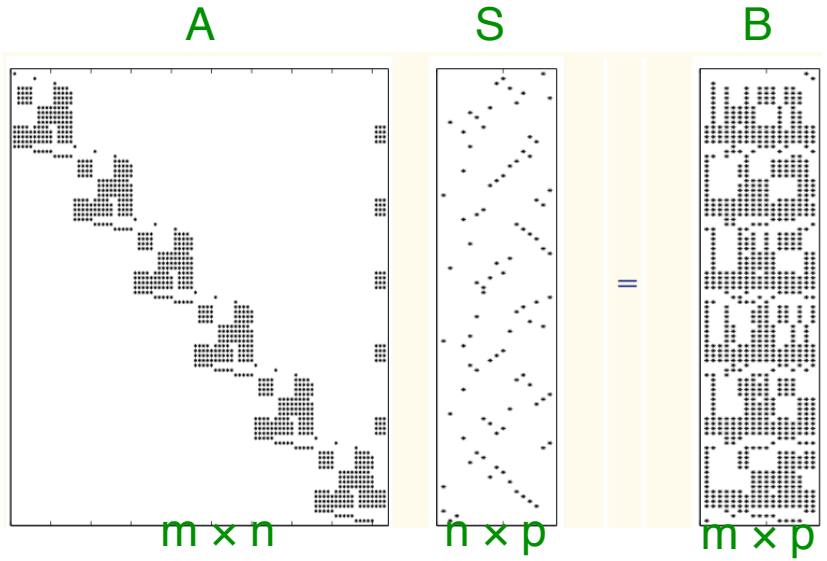
$$\begin{bmatrix} j_{11} & j_{12} & 0 & 0 & j_{15} \\ 0 & 0 & j_{23} & 0 & 0 \\ 0 & j_{32} & j_{33} & j_{34} & 0 \\ j_{41} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & j_{54} & j_{55} \end{bmatrix} \quad \begin{bmatrix} j_{11} & j_{12} & 0 & 0 & j_{15} \\ 0 & 0 & j_{23} & 0 & 0 \\ 0 & j_{32} & j_{33} & j_{34} & 0 \\ j_{41} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & j_{54} & j_{55} \end{bmatrix} \quad \begin{bmatrix} j_{11} & j_{12} & j_{15} \\ 0 & 0 & j_{23} \\ j_{34} & j_{32} & j_{33} \\ j_{41} & 0 & 0 \\ j_{54} & 0 & j_{55} \end{bmatrix}$$

- Structurally orthogonal columns do not have non-zero elements in the same row
- Grouping them allows them to be calculated together

Using Coloring Models in Derivative Computation

4-step procedure:

- *S1: Determine the sparsity structure of A* **(SparsLinC Library)**
- *S2: Obtain a seed matrix S by coloring the graph of A* **(ColPack)**
- *S3: Compute a compressed matrix B=AS* **(ColPack, ADIC2)**
- *S4: Recover entries of A from B* **(ColPack)**



Example of ADIC2 Generated Code:

```
1. Log into the VM.

2. cd ~/ADIC/

3. source setenv_inst.sh              -- Sets required environment variables

4. cd examples/arith_trig1_sparse/-- Go into the example directory

5. ls                                 -- What files do you see?

6. make clean & make                  -- Generate output and compile it

7. ls *.c *.h *.dat                   -- What are the new files?
                                         head.cn.xb.pp.c
                                         ad_grad_saxpy-n_sparse.h
                                         seed_matrix.dat
                                         sparsity_pattern.dat
                                         vertex_colors.dat
                                         adic_gradvec_length.h
```

Example of ADIC2 Generated Code : Driver

```
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x,int n);

int main()
{
  //Declarations & Initialization --Omitted

  // Set independent variables --Omitted

  // Initialize the value of the independent variable ad_x --Omitted

  // Invoke AD function --Omitted

  //Invoke ColPack compression
  CSR_NOGRAD * csrobject = new CSR_NOGRAD(ARRAY_SIZE);
  csrobject->addToJacobian(ARRAY_SIZE, ad_y);
  std::list<std::set<int> > csi_SparsityPattern;
  csi_SparsityPattern = csrobject->valsetlist;
  ColPack::BipartiteGraphPartialColoringInterface * g;
  g = new ColPack::BipartiteGraphPartialColoringInterface
    (SRC_MEM_ADIC, &csi_SparsityPattern, csrobject->column_count);
  g->PartialDistanceTwoColoring("NATURAL","COLUMN_PARTIAL_DISTANCE_TWO");
  double*** dp3_Seed = new double**;
  int rowcount, colcount;
  (*dp3_Seed) = g->GetSeedMatrix(&rowcount, &colcount);

  //Generate output files
  printmat("Seed Matrix", rowcount, colcount, *dp3_Seed,"seed_matrix.dat");
  WritePatternToFile("sparsity_pattern.dat", csi_SparsityPattern);
  vector<int> vi_RightVertexColors;
  g->GetRightVertexColors(vi_RightVertexColors);
  WriteVertexColorsToFile("vertex_colors.dat", vi_RightVertexColors);

  //Finalize
  ADIC_Finalize(colcount);
  return 0;
}
```

x[0].indexSet	0
x[1].indexSet	1
y[0].indexSet	Empty
y[1].indexSet	Empty
x[0].val	0.5
x[1].val	0.6
y[0].val	0.0
y[1].val	0.0

- **Step1:** SparsLinC – a library that implements sparsity pattern detection
  - We use the same output as before
  - Modified versions of the macros that maintain sets of non-zero indices elements of each row in the Jacobian
  - No derivative computation is performed
  - Various techniques have been explored by other researchers

```
class SparseDeriv
  {
    public:
      SparseDeriv() {}
      ~SparseDeriv() {}
      std::set<int> indexSet;
  };

typedef SparseDeriv DERIV_TYPE;
```

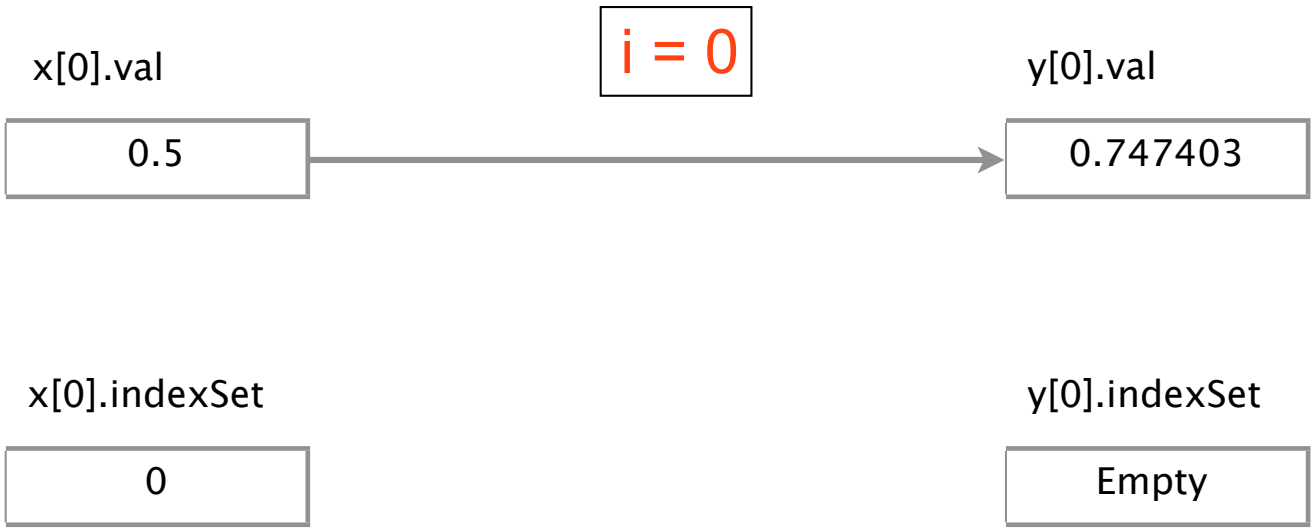
ad_grad_saxpy-n_sparse.h

```
void ADIC_Sax_3(  double a1, DERIV_TYPE &x1,
                  double a2, DERIV_TYPE &x2,
                  double a3, DERIV_TYPE &x3, DERIV_TYPE &tgt) {
  std::set<int>::iterator srcSetIter;
  tgt->indexSet =x1->indexSet;
  srcSetIter = x2->indexSet.begin();
  for (; srcSetIter != x2->indexSet.end(); srcSetIter++) {
    tgt->indexSet.insert(*srcSetIter);
  }
  srcSetIter = x3->indexSet.begin();
  for (; srcSetIter != x3->indexSet.end(); srcSetIter++) {
    tgt->indexSet.insert(*srcSetIter);
  }
}
```

- Points to consider
  - Does your sparsity pattern change during execution?

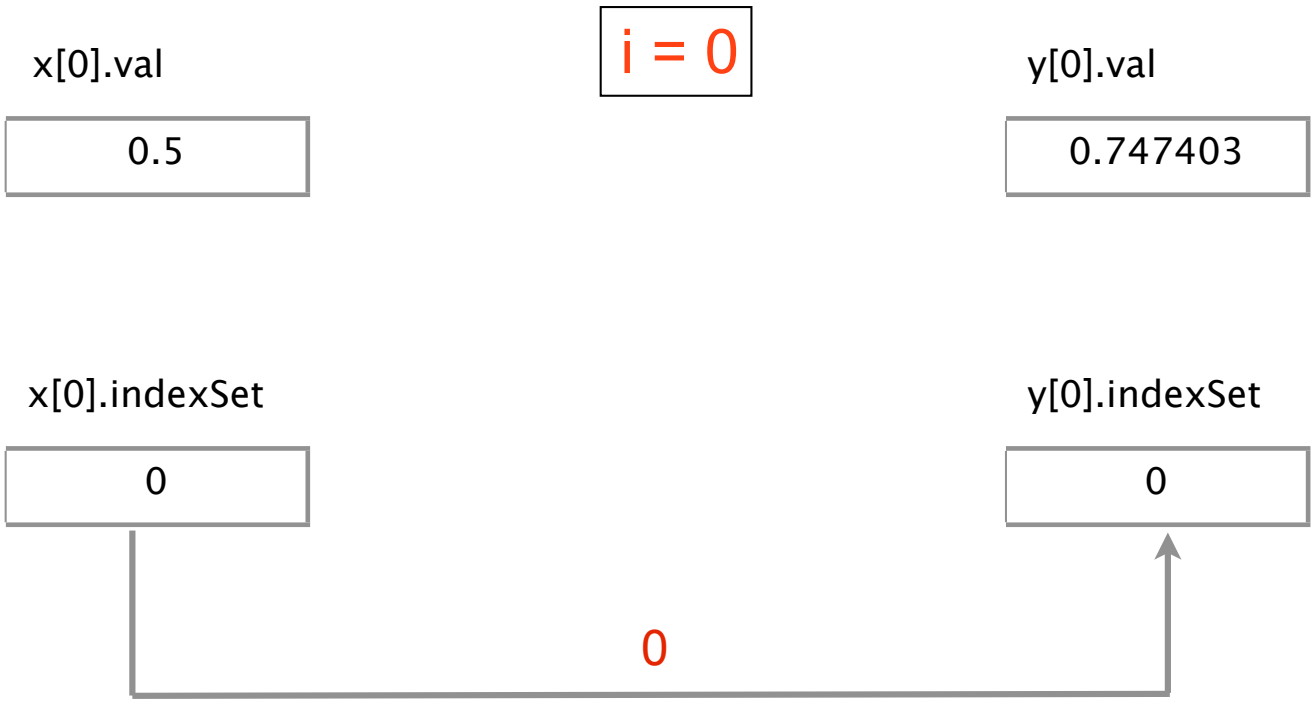
What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



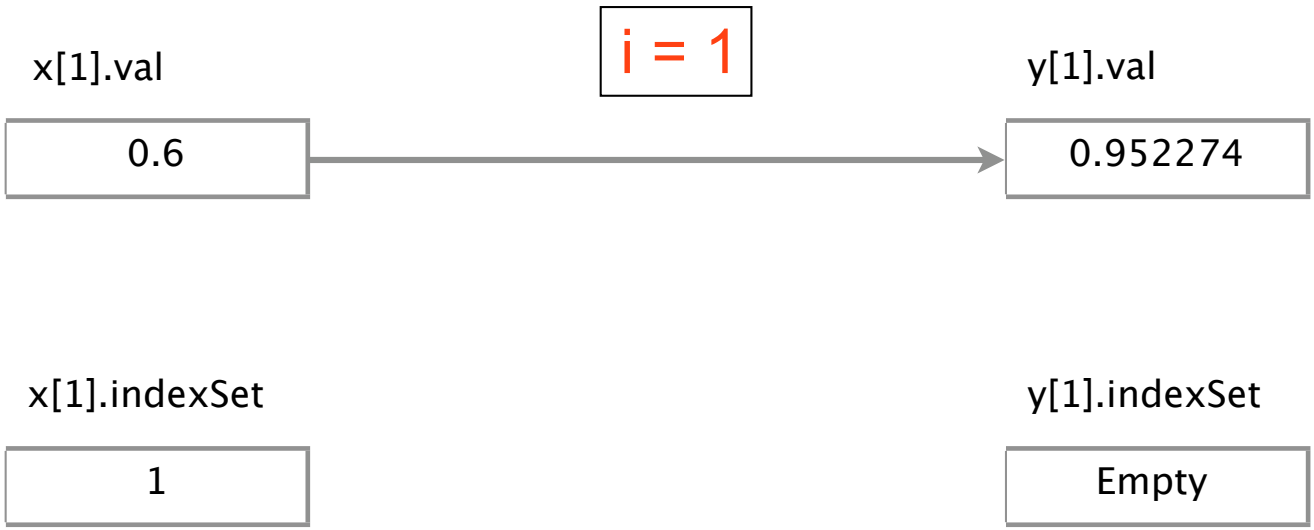
What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



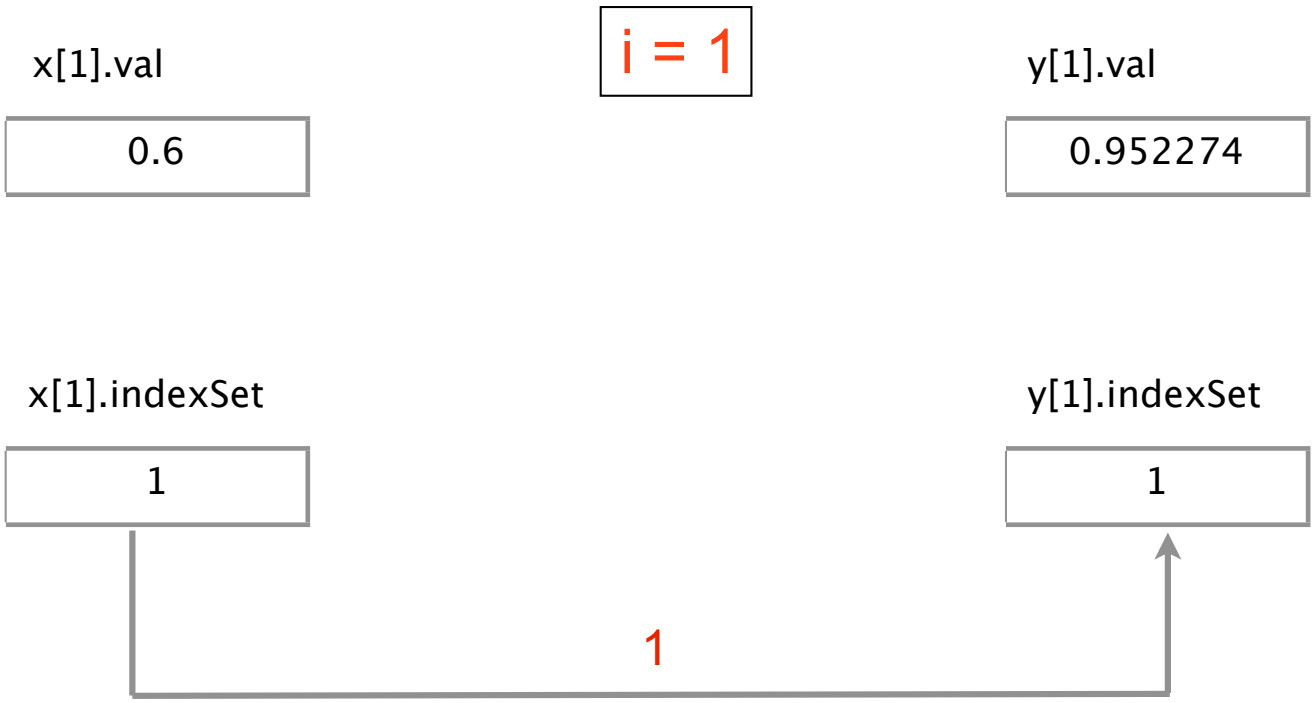
What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```

x[0].val	0.5
x[1].val	0.6

y[0].val	0.0
y[1].val	0.0

x[0].indexSet	0
x[1].indexSet	1

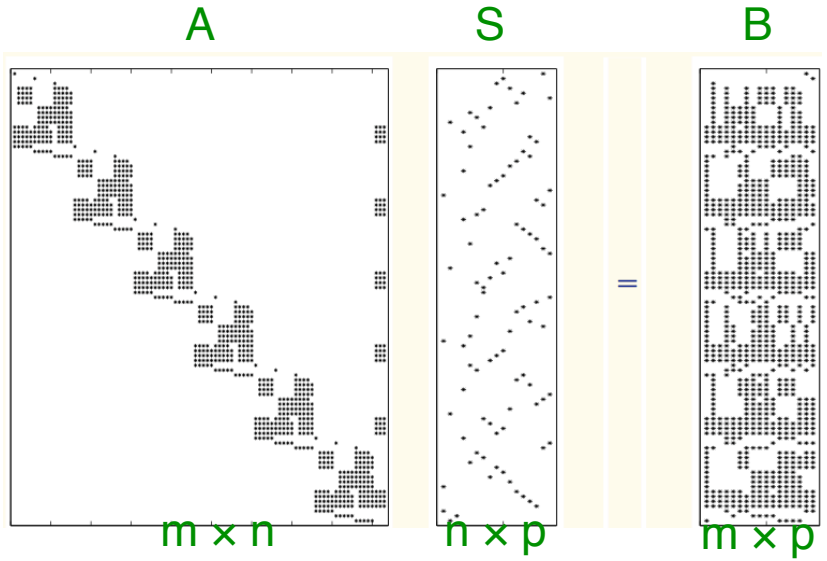
y[0].indexSet	0
y[1].indexSet	1

`sparsity_pattern.dat`

Using Coloring Models in Derivative Computation

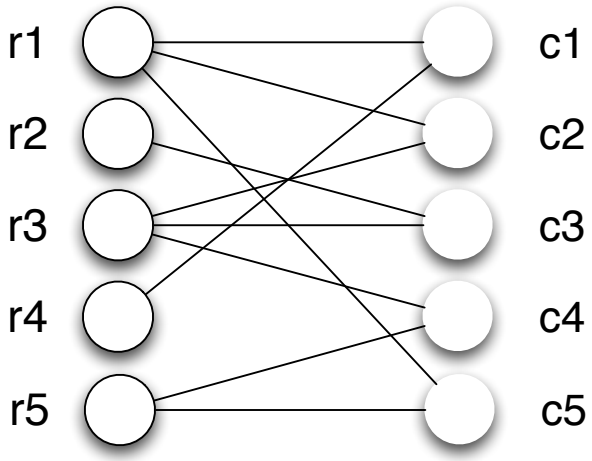
4-step procedure:

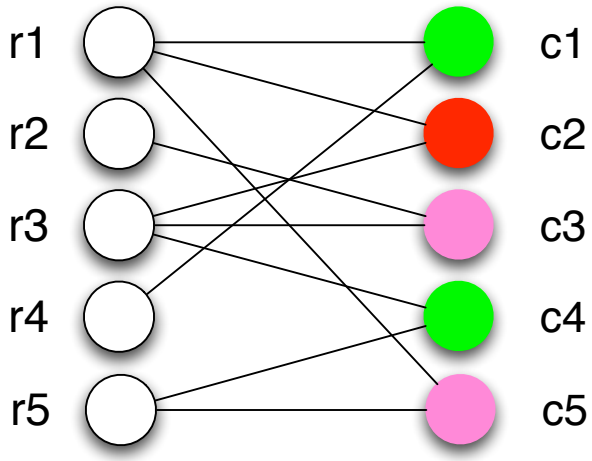
- ~~*S1: Determine the sparsity structure of A*~~ **~~(SparsLinC Library)~~**
- *S2: Obtain a seed matrix S by coloring the graph of A* **(ColPack)**
- *S3: Compute a compressed matrix B=AS* **(ColPack, ADIC2)**
- *S4: Recover entries of A from B* **(ColPack)**



Using Coloring Models in Derivative Computation

$$\begin{bmatrix} j_{11} & j_{12} & 0 & 0 & j_{15} \\ 0 & 0 & j_{23} & 0 & 0 \\ 0 & j_{32} & j_{33} & j_{34} & 0 \\ j_{41} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & j_{54} & j_{55} \end{bmatrix}$$





`vertex_colors.dat`

- Step 2: Partitioning Of The Jacobian Into Structurally Orthogonal Columns
  - Form a bipartite graph
  - Color the column nodes using distance-2 coloring
  - Obtain a seed matrix: `seed_matrix.dat`
  - Generate `adic_gradvec_length.h`

Using Coloring Models in Derivative Computation

- Step 3: Computation of the sparse Jacobian
  - Same as earlier, but the length of the grad array is limited to the number of colors

```
typedef struct {
  double val;
  double grad[NUMBER_OF_COLORS];
} DERIV_TYPE;
```

- Step 4: Extraction of the original Jacobian
  - Colpack provides functions for this

Example of ADIC2 Generated Code:

```
1. Log into the VM.

2. cd ~/ADIC/

3. source setenv_inst.sh          -- Sets required environment variables

4. cd examples/arith_trig1_seed/  -- Go into the example directory

5. ls                             -- What files do you see?

6. cp ../arith_trig1_seed/*.dat ../arith_trig1_seed/adic_gradvec_length.h ./
                                  -- Obtain generated files

6. make clean & make              -- Generate output and compile it

7. ./driver                       -- Obtain output
```

Example of ADIC2 Generated Code : Driver

```
#define ARRAY_SIZE 2
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x,int n);
int main()
{
  //Declarations & Initialization --Omitted

  //Read inputfiles on compressed format
  int * x_vertexcolors;
  ADIC_ReadVertexColorsFromFile("vertex_colors.dat", &x_vertexcolors);
  double ** x_seedmatrix;
  int x_seed_matrix_row_count, x_seed_matrix_column_count;
  ADIC_GetSeedMatrixFromFile("seed_matrix.dat", &x_seedmatrix,
              &x_seed_matrix_row_count,  &x_seed_matrix_column_count);
  int **x_sparsitypattern;
  ADIC_ReadPatternFromFile("sparsity_pattern.dat", &x_sparsitypattern);


  // Set indpendent variables
  ADIC_SetForwardMode();
  ADIC_SetDepArray(ad_y, ARRAY_SIZE);
  for (i=0; i<ADIC_GRADVEC_LENGTH; i++){
    __ADIC_IncrShadowVar();
  }
  ADIC_SetIndepArrayFromSeed(ad_x, x_seedmatrix, x_seed_matrix_row_count,
                                        x_seed_matrix_column_count);
  ADIC_SetIndepDone();

  // Initialize the value of the independent variable ad_x --Omitted

  // Invoke AD function --Omitted

  double** yx_fulljacobian =  (double**) malloc(ARRAY_SIZE * sizeof(double*));
  for (i=0; i<ARRAY_SIZE; i++) {
    yx_fulljacobian[i] = (double*) calloc(ARRAY_SIZE, sizeof(double));
  }
  ADIC_RecoverFullJacobian(ad_y, yx_fulljacobian, x_sparsitypattern,
                                    x_seed_matrix_row_count, x_vertexcolors);

  //Free data structures --Omitted
}
```

y[0].grad	0.0
y[1].grad	0.0
x[0].grad	1.0
x[1].grad	1.0
x[0].val	0.5
x[1].val	0.6
y[0].val	0.0
y[1].val	0.0

What Actually Happens During Execution?

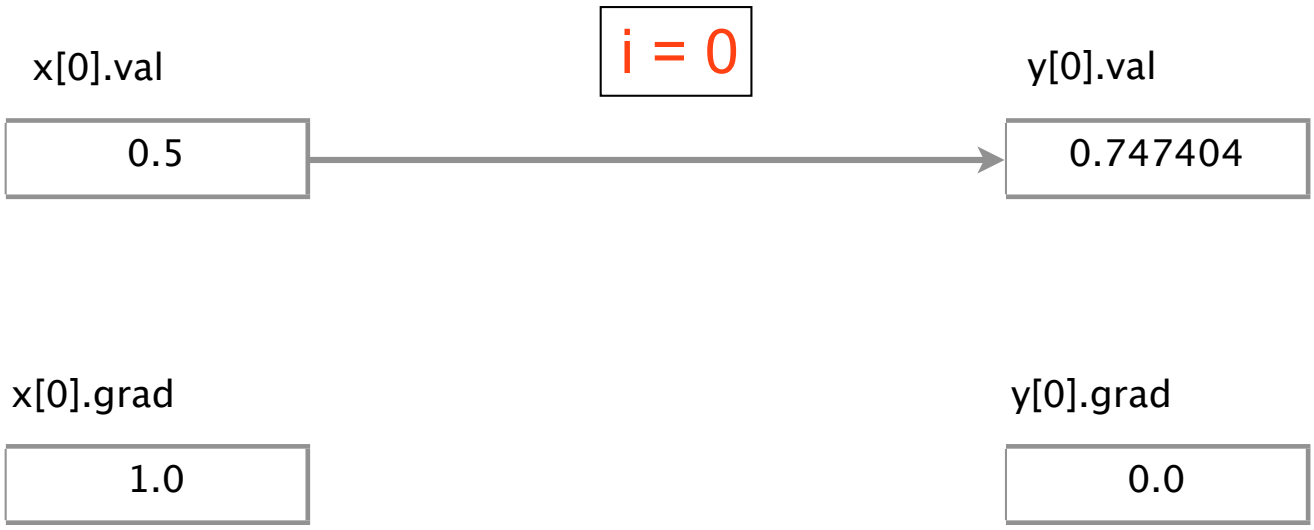
```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```

x[0].val	0.5	y[0].val	0.0
x[1].val	0.6	y[1].val	0.0

x[0].grad	1.0	y[0].grad	0.0
x[1].grad	1.0	y[1].grad	0.0

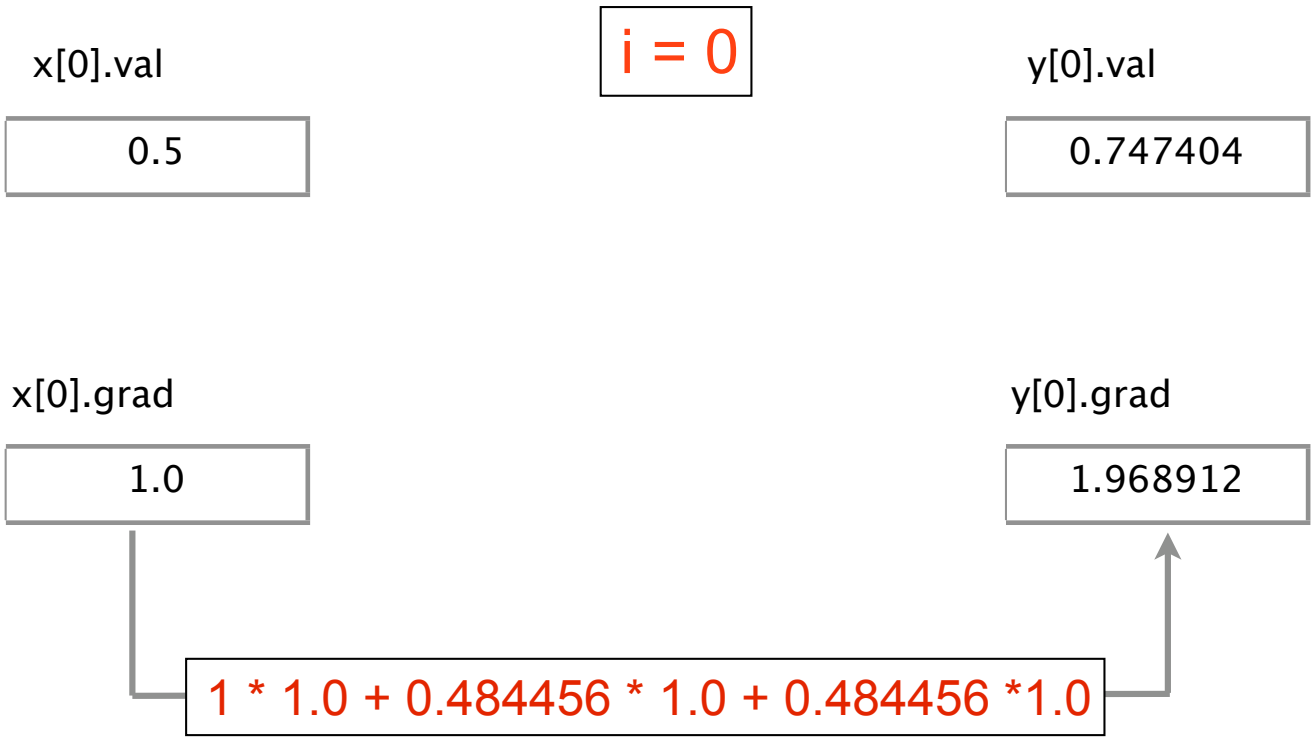
What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



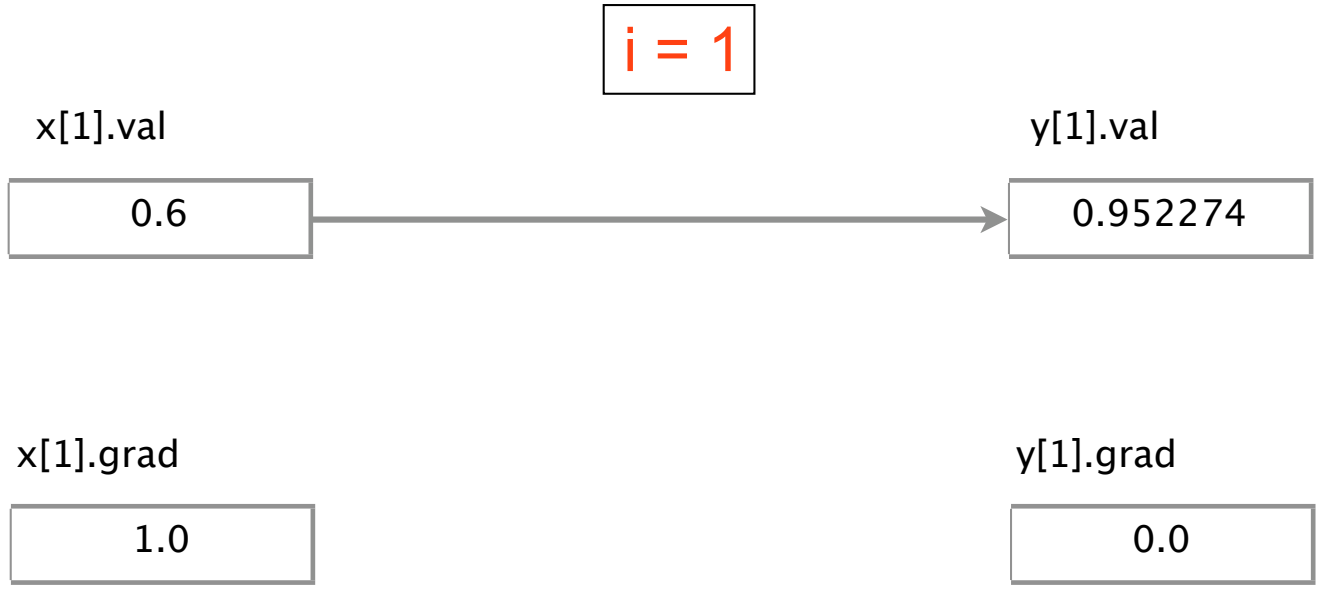
What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



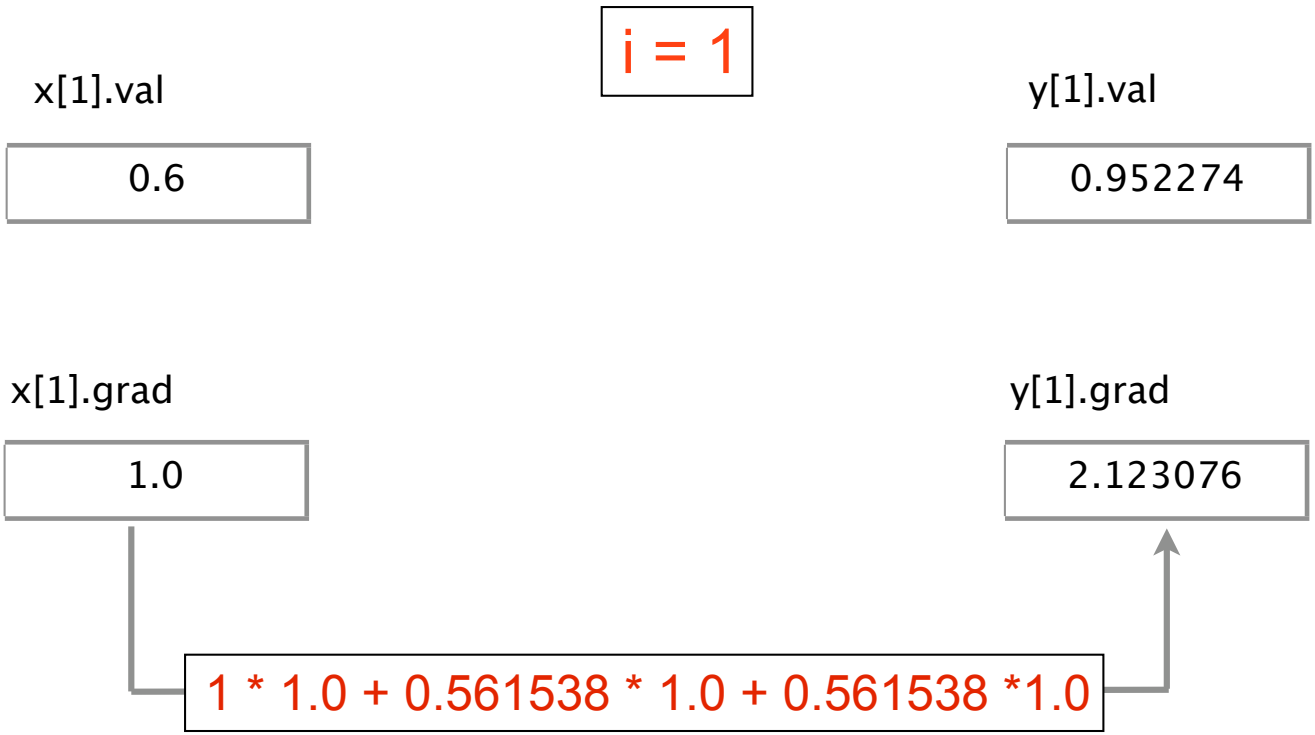
What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```



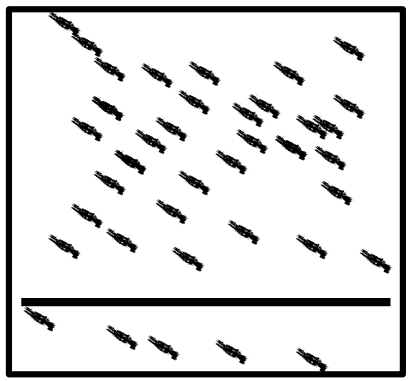
What Actually Happens During Execution?

```
#include "ad_types.h"
void ad_mini1(DERIV_TYPE *y,DERIV_TYPE *x)
{
  int i;
  double temp0, temp1, temp2, temp3;
  for (i = 0; i < 2; i = (i + 1)) {
    temp0 = x[i].val;
    temp1 = temp0 * temp0;
    y[i].val = temp0 + sin(temp1);
    temp2 = cos(temp1);
    temp3 = temp0 * temp2 ;
    ADIC_Sax_3(1, x[i].grad, temp3, x[i].grad, temp3, x[i].grad, y[i].grad);
  }
}
```

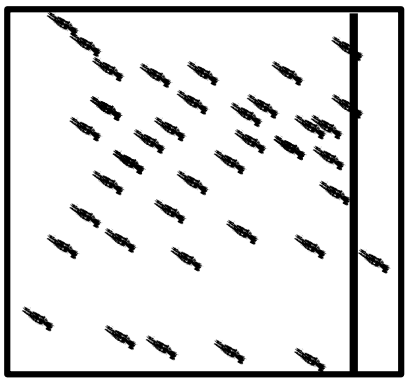


Other modes

- Select your mode based on your sparsity pattern.
  - Forward mode
  - Reverse mode
  - Mixed mode



Bad for forward mode



Bad for reverse mode

Partial separability

- Partial separability: A structure often exhibited by large optimization problems (e.g., least squares problems)
- Given

$$f : \mathbb{R}^n \mapsto \mathbb{R}$$

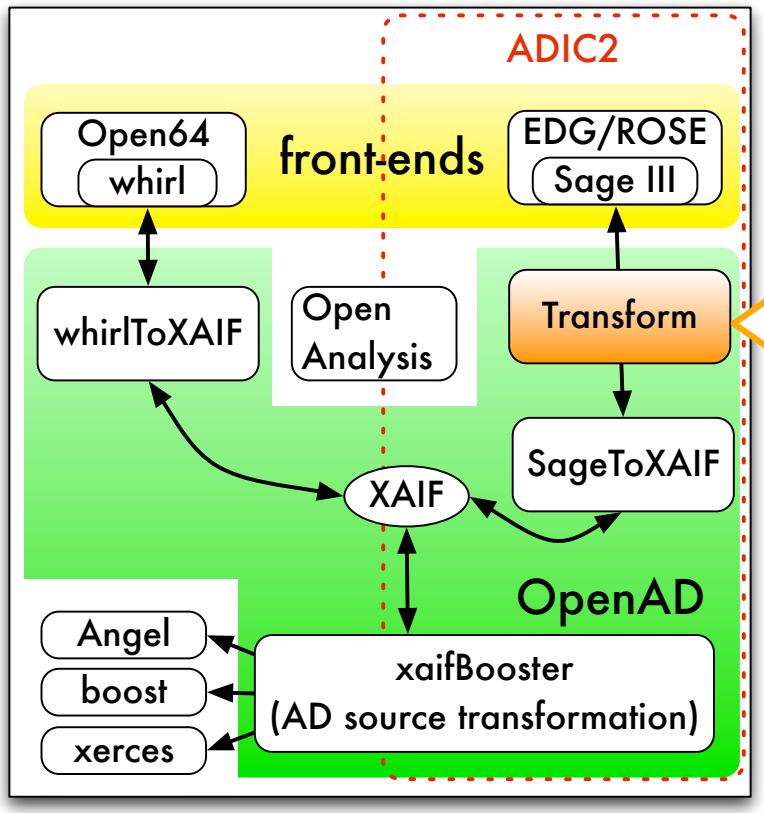
$$f(x) = \sum_{i=1}^{m} f_i(x)$$

where f_i depends on $p_i \ll n$ variables.

- Problem: The gradient of the partially separable function can be dense even if the gradient of each elemental is sparse.

Partial separability

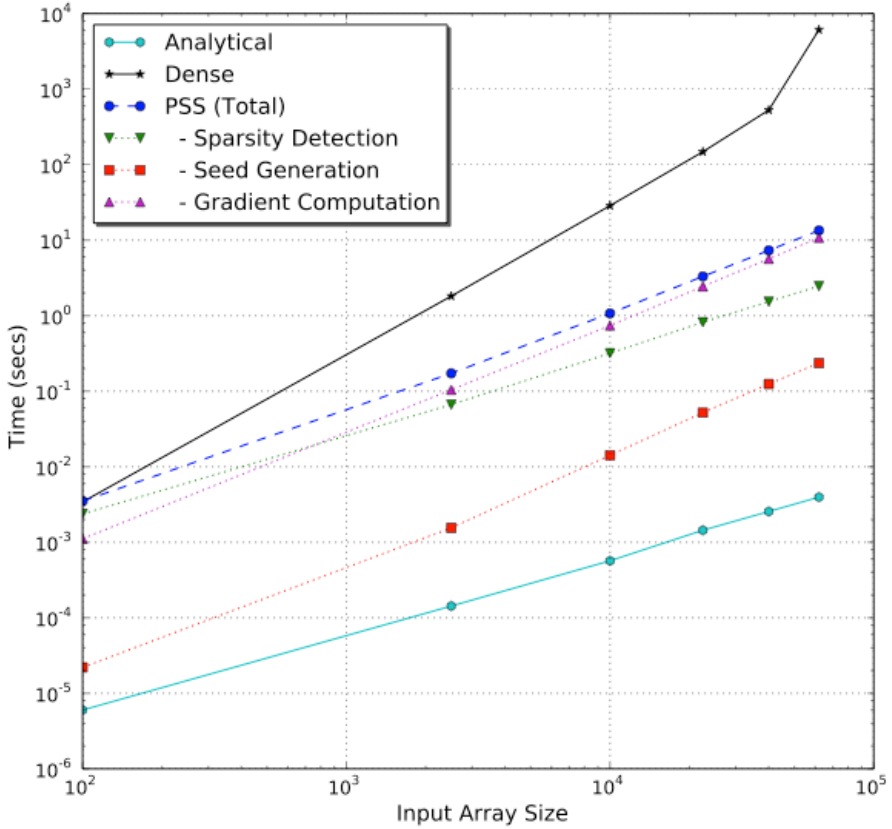
- Problem: The gradient of the partially separable function can be dense even if the gradient of the elementals is sparse.
- Solution: First compute the much smaller gradients of the elementals, then assemble the full gradient of f.

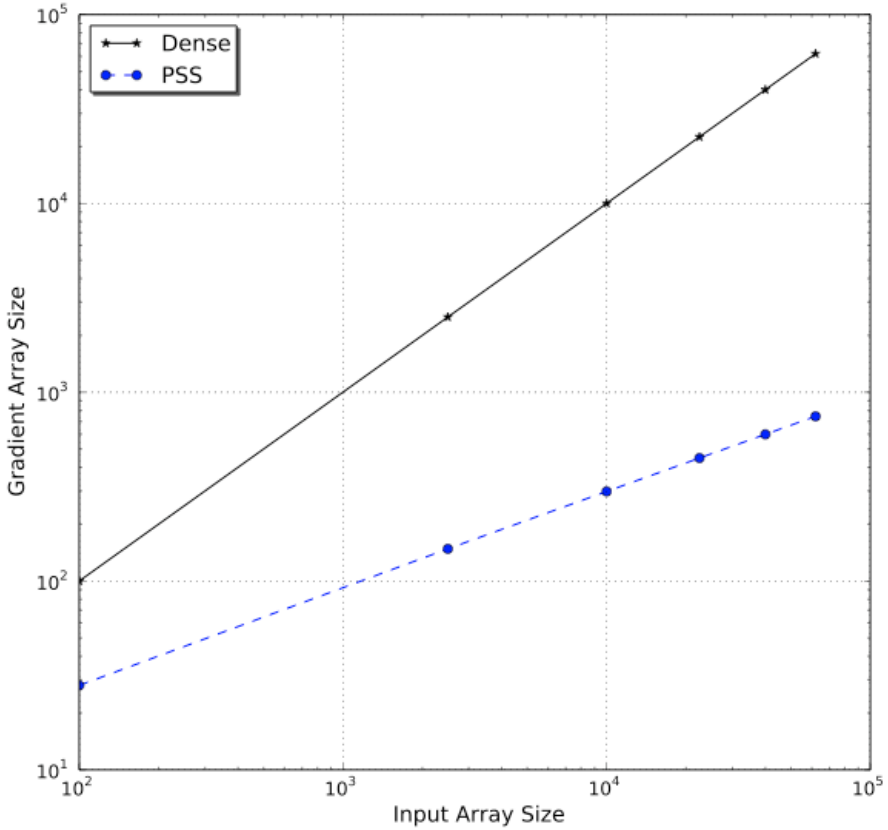


1. Identification of partial separability
2. Scalar expansion of elemental functions within loops and creation of a summation function
3. Generation of elemental initialization loop

Performance

- Result: up to 1,000x time reduction over dense for MINPACK2's elastic-plastic torsion problem.





Conclusions

- Showed how sparsity support has been implemented in ADIC2 for forward mode
- Similar technique exists in ADOL-C
- ADIC2
  - https://trac.mcs.anl.gov/projects/ADIC
- ColPack
  - http://www.cscapes.org/coloringpage/
- Automatic Differentiation
  - http://www.autodiff.org/
  - www.mcs.anl.gov/autodiff/

Argonne NATIONAL LABORATORY

Forward mode source transformation AD using ADIC2 : A larger example

Sri Hari Krishna Narayanan and Jean Utke

Laboratory for Advanced Numerical Simulations

Mathematics and Computer Science Division

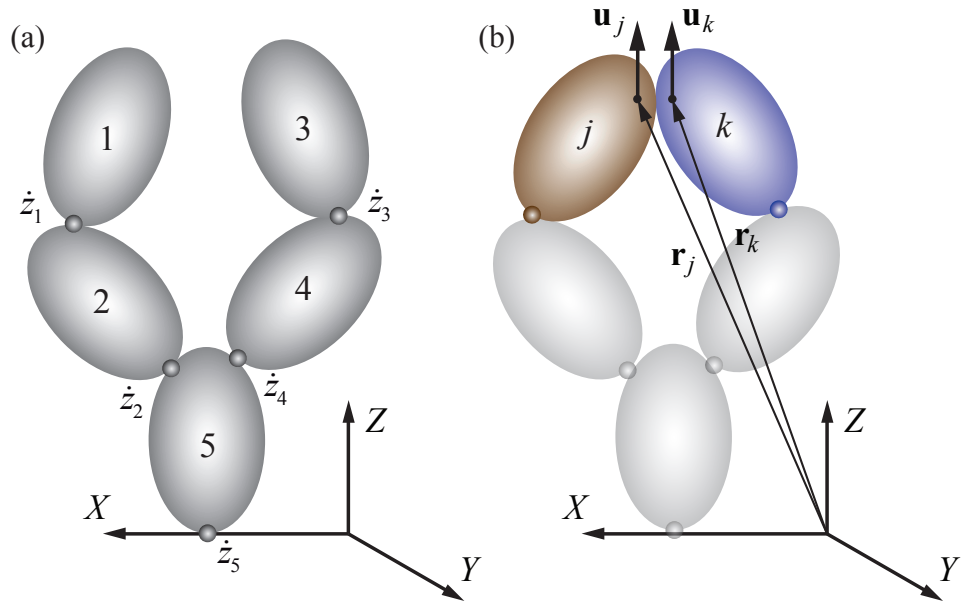
Application by Alfonso Callejo Goena

UPM, Spain

U.S. DEPARTMENT OF ENERGY

Multibody Systems

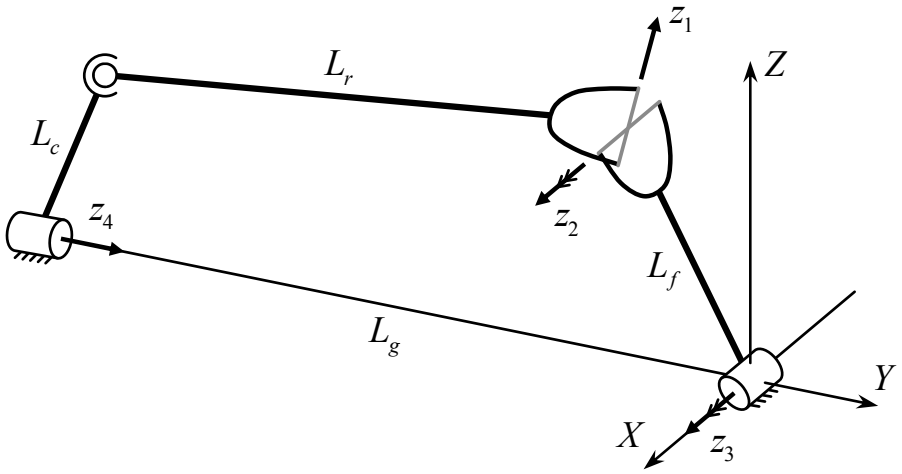
- Mechanical systems consisting of rigid or flexible bodies, connected via imperfect kinematic joints, and subject to various forces
- Examples: Robots, vehicles, machinery, wind turbines
- Their behavior is typically simulated.
- It has been an active field for about 35 years

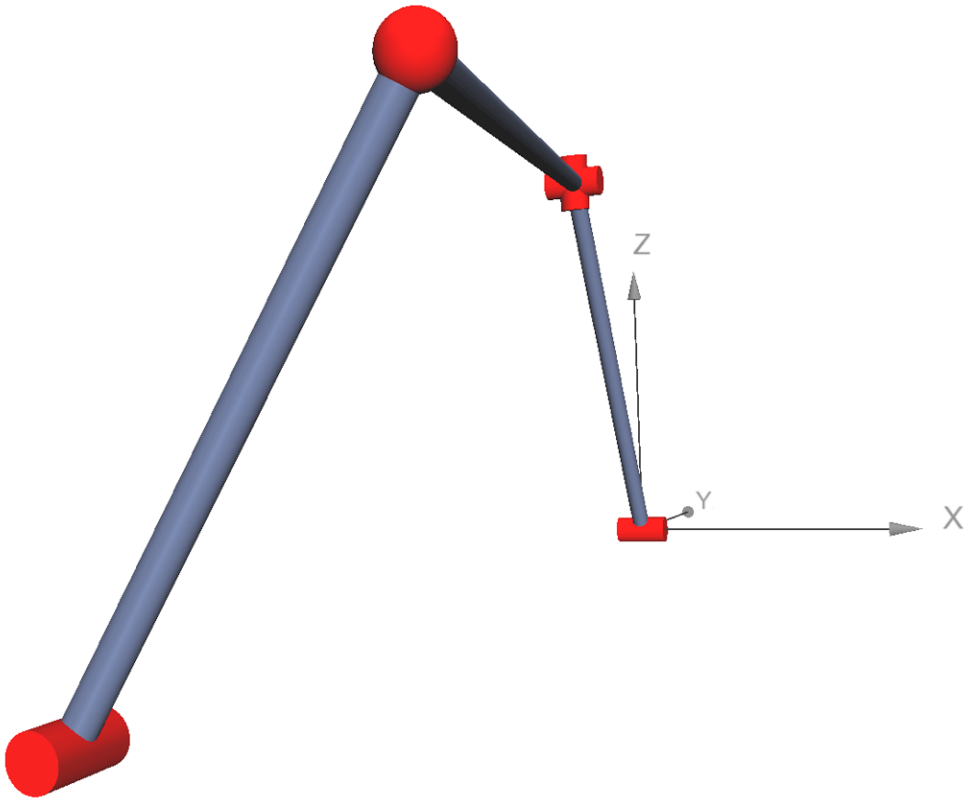


$$\mathbf{M}^{\Sigma}_{d,j+1}\mathbf{z}_{j+1}+\frac{h^2}{4}\mathbf{\Phi}^{T}_{\mathbf{z}_{j+1}}\alpha\mathbf{\Phi}_{j+1}-\frac{h^2}{4}\mathbf{Q}^{\Sigma}_{d,j+1}+\frac{h^2}{4}\mathbf{M}^{\Sigma}_{d,j+1}\hat{\ddot{\mathbf{z}}}_j=\mathbf{0}$$

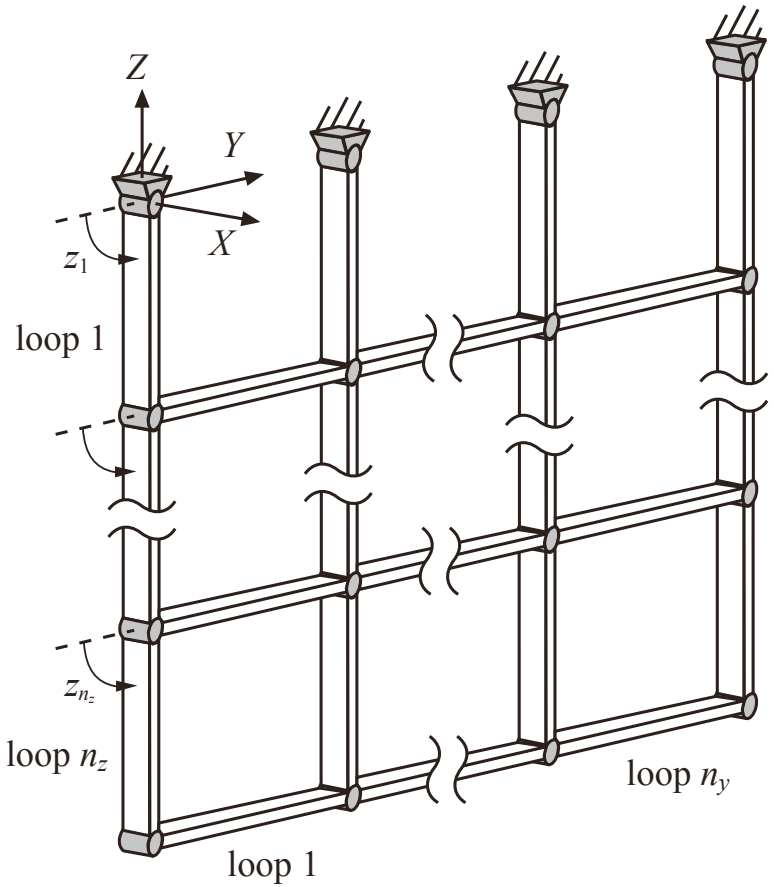
- Nonlinear system of equations solved by Newton–Raphson.
- Jacobian is used for 3 iterations.
  - Used finite differences prior to AD

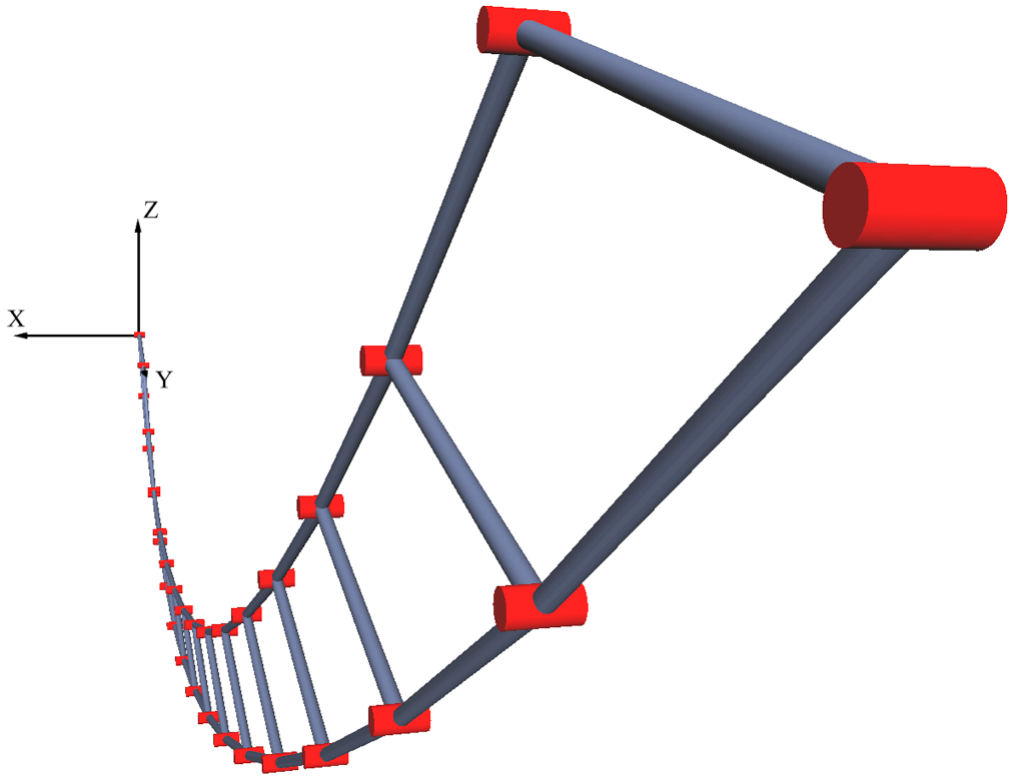
MBS 1: Spatial four-bar mechanism





MBS 2: Multiple four-bar linkage





MBS 3: Coach dynamic maneuver

Recipe for using ADIC2

- Step 1: Identification of opportunity
  - Identify the process that needs derivatives
  - Identify the function to be called that has be be differentiated
  - Identify the independent and dependent variables

Recipe for using ADIC2

- Step 2: If your project contains multiple files, know what each file contains

```
SRC_MULTIPLEFOURBAR = constr3.c
evalExtForcesC.c
evaluateFipzzpC.c
IMP_energyBalance.c
IMP_evaluateMjkC.c
IMP_integradorTRC.c
IMP_jacobian.c
IMP_jacobian_ADIC.c
IMP_main.c
IMP_recAccQacc3C.c
IMP_recPos3C.c
IMP_RTDynTRC.c
recVel3C.c
solveLU.c
Standalone_tr_mbsOptions_Read.c
Standalone_tr_openClosedLoops_Read.c
Standalone_tr_Tnode_Read.c
Standalone_tr_Trod_Read.c
StandaloneReadFunctions.c
utilfunc.c

multipleFourBarDrCoor.c
multipleFourBarUserForces.c
```

```
SRC_SPATIALFOURBAR = constr3.c
evalExtForcesC.c
evaluateFipzzpC.c
IMP_energyBalance.c
IMP_evaluateMjkC.c
IMP_integradorTRC.c
IMP_jacobian.c
IMP_jacobian_ADIC.c
IMP_main.c
IMP_recAccQacc3C.c
IMP_recPos3C.c
IMP_RTDynTRC.c
recVel3C.c
solveLU.c
Standalone_tr_mbsOptions_Read.c
Standalone_tr_openClosedLoops_Read.c
Standalone_tr_Tnode_Read.c
Standalone_tr_Trod_Read.c
StandaloneReadFunctions.c
utilfunc.c

spatialFourBarDrCoor.c
spatialFourBarUserForces.c
```

```
SRC_BUSORIGINAL = constr3.c
evalExtForcesC.c
evaluateFipzzpC.c
IMP_energyBalance.c
IMP_evaluateMjkC.c
IMP_integradorTRC.c
IMP_jacobian.c
IMP_jacobian_ADIC.c
IMP_main.c
IMP_recAccQacc3C.c
IMP_recPos3C.c
IMP_RTDynTRC.c
recVel3C.c
solveLU.c
Standalone_tr_mbsOptions_Read.c
Standalone_tr_openClosedLoops_Read.c
Standalone_tr_Tnode_Read.c
Standalone_tr_Trod_Read.c
StandaloneReadFunctions.c
utilfunc.c

busDrivenCoordinateVehicle.c
bus_force_damper.c
bus_force_spring.c
busMagicFormulaL2002.c
busManeuverLaneChangeSpline.c
busPacejka.c
busTireForces.c
busUserForces.c
busWheelTorques.c
```


Recipe for using ADIC2

- Step 3: Separate out active and inactive files
  - Separate ‘mixed’ files
  - Ensure that everything builds

```
SRC_MULTIPLEFOURBAR =
utilfunc_active.c
IMP_RTDynTRC.c
IMP_recPos3C.c
recVel3C.c
evalExtForcesC.c
IMP_evaluateMjkC.c
IMP_recAccQacc3C.c

multipleFourBarUserForces.c
```

```
SRC_SPATIALFOURBAR =
utilfunc_active.c
IMP_RTDynTRC.c
IMP_recPos3C.c
recVel3C.c
evalExtForcesC.c
IMP_evaluateMjkC.c
IMP_recAccQacc3C.c

spatialFourBarUserForces.c
```

```
SRC_BUSORIGINAL =
utilfunc_active.c
IMP_RTDynTRC.c
IMP_recPos3C.c
recVel3C.c
evalExtForcesC.c
IMP_evaluateMjkC.c
IMP_recAccQacc3C.c

bus_force_damper.c
bus_force_spring.c
busMagicFormulaL2002.c
busPacejka.c
busTireForces.c
busUserForces.c
busWheelTorques.c
```

Recipe for using ADIC2

- Step 4: Write any stubs that are needed
  - Put guards around them
  - Ensure that everything builds

```
double ** createMatrix(int m, int n){ /* ACTIVE */
  int i;
  double **matrix, *mat;
  matrix = (double**)calloc(m, sizeof(double *));
  matrix[0] = mat = (double*)calloc(m*n, sizeof(double));
  for (i=1; i<m; i++){
    matrix[i] = mat+n*i;
  }
  return matrix;
}
```

```
void createMatrix(double** input, int m, int n){
}
```

```
#ifdef DIFFERENTIATION
  int aDummyVariableToHelpAD_2 = 1;
 createVector(RdtQacc, nocJoints);
 createVector(RdtPacc, nocJoints);
 createVector(jointB, 6*nocJoints);
 createMatrix(jointMacc, 6*nocJoints,6);
#else
  int aDummyVariableToHelpAD_2 = 1;
 RdtQacc   = createVector(nocJoints);
 RdtPacc   = createVector(nocJoints);
 jointB    = createVector(6*nocJoints);
 jointMacc = createMatrix(6*nocJoints,6);
#endif
```

Recipe for using ADIC2

- Step 5: Driver generation
  - Identify the function to be called that has be be differentiated
  - Wrap that in a driver
    - Write interfaces to copy any data that needs to be copied between passive and active variables.
    - Include runtime headers in the driver

- Step 6: Invocation of ADIC2
  - Concatenate active files
  - Invoke ADIC2 on the concatenated files
  - File local variables are a problem

- Step 7: Invocation of ADIC2
  - Postprocess the differentiated output (by hand for now)
  - So far this involves removing the differentiated stubs.

Recipe for using ADIC2

- Step 8: Differentiate header Files
  - Rose does not handle header files in a straightforward way
    - Have to make them into a source file
  - Differentiated headers and original headers might have to co-exist!
    - Differentiated structure definitions, function headers.

```
cat cnode.h tmp_main.c > cnode_header.c
cat IMP_common.h tmp_main.c > IMP_common_header.c
cat solveLU.h tmp_main.c > solveLU_header.c
cat utilfunc_active.h tmp_main.c > utilfunc_active_header.c


cat busUserForces.h tmp_main.c > busUserForces_header.c
cat busPacejkaCoefficients.h tmp_main.c > busPacejkaCoefficients_header.c
```

```
#ifndef _IMPCOMMON_INCLUDE_
#define _IMPCOMMON_INCLUDE_
```

```
#ifndef _ADIC_IMPCOMMON_INCLUDE_
#define _ADIC_IMPCOMMON_INCLUDE_
```

- One approach is to pre-process the header files

Recipe for using ADIC2

- Step 9: Write a makefile the builds everything.
  - Best approach is to adopt an existing makefile
- Select the appropriate runtime library
  - Forward, Reverse, Sparse (more on this later)

Recipe for using ADIC2

- Possible pitfalls
  - You may need too much stack space

```
typedef struct {
  double val;
  double grad[ADIC_GRADVEC_LENGTH];
} DERIV_TYPE;
```

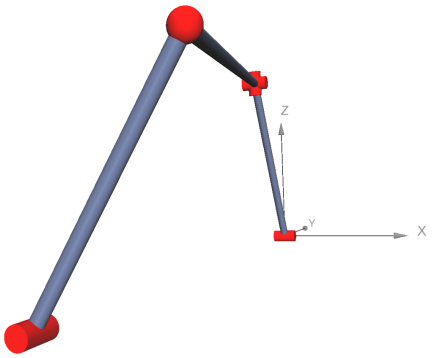
- Increase it to the maximum possible

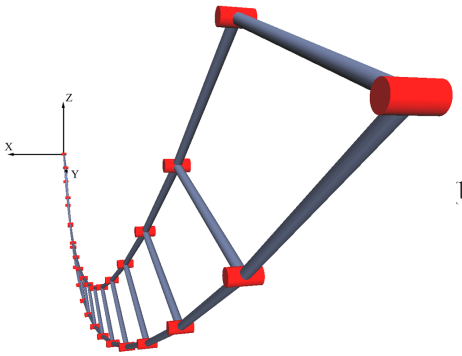
```
struct rlimit rl;
int result;

result = getrlimit(RLIMIT_STACK, &rl);
if (result == 0)  {
  rl.rlim_cur = rl.rlim_max;
  result = setrlimit(RLIMIT_STACK, &rl);
  if (result != 0) {
    fprintf(stderr, "setrlimit returned result = %dn", result);
  }
} else
  fprintf(stderr, "getrlimit returned result = %dn", result);
```

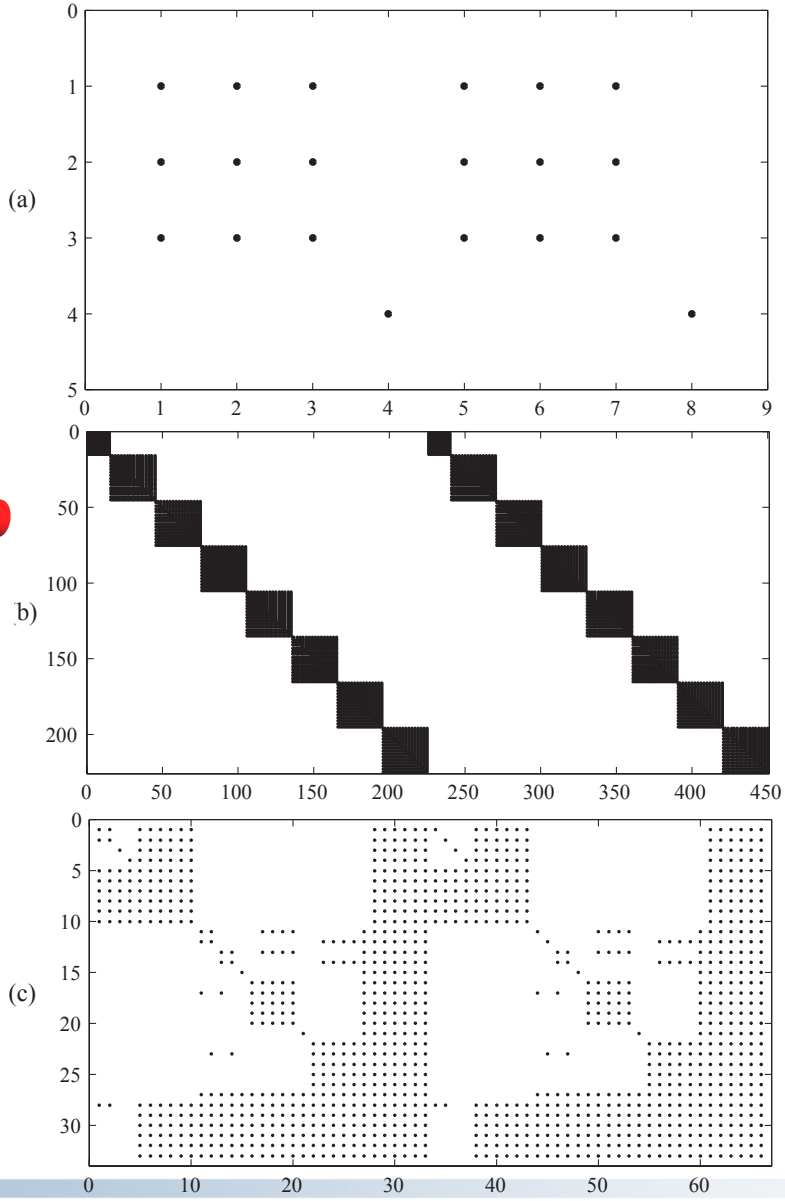
How to run the multibody code (homework?)

```
1. Log into the VM.
2. cd ~/ADIC/
3. source setenv_inst.sh              -- Sets required environment variables
4. cd examples/multibody/SplitFiles/adic_output
                                      -- Go into the example directory
5. make clean
6. make multiplefourbar1_20           -- Compile generated files
7. ./multiplefourbar1_20 multipleFourBar1-20 ../bin/multipleFourBar1-20/
                                      -- Run the code
   ./spatialfourbar1   spatialFourBar1   ../bin/spatialFourBar1/
   ./spatialfourbar10  spatialFourBar10  ../bin/spatialFourBar10/
   ./spatialfourbar20  spatialFourBar20  ../bin/spatialFourBar20/
   ./multiplefourbar1_1  multipleFourBar1-1  ../bin/multipleFourBar1-1/
   ./multiplefourbar1_10 multipleFourBar1-10 ../bin/multipleFourBar1-10/
   ./multiplefourbar1_20 multipleFourBar1-20 ../bin/multipleFourBar1-20/
   ./multiplefourbar4_1  multipleFourBar4-1  ../bin/multipleFourBar4-1/
   ./multiplefourbar4_10 multipleFourBar4-10 ../bin/multipleFourBar4-10/
   ./multiplefourbar4_20 multipleFourBar4-20 ../bin/multipleFourBar4-20/
   ./multiplefourbar7_1  multipleFourBar7-1  ../bin/multipleFourBar7-1/
   ./multiplefourbar7_10 multipleFourBar7-10 ../bin/multipleFourBar7-10/
   ./multiplefourbar7_20 multipleFourBar7-20 ../bin/multipleFourBar7-20/
   ./busoriginal1   busOriginal1   ../bin/busOriginal1/
   ./busoriginal10  busOriginal10  ../bin/busOriginal10/
   ./busoriginal20  busOriginal20  ../bin/busOriginal20/
```

Sparsity pattern that was detected for multibody application







System	Indeps.	Deps.	Nonzeroes	Colors
Spatial four-bar	8	4	20	6
1 × 15 four-bar	90	45	1410	60
4 × 15 four-bar	270	135	4290	60
7 × 15 four-bar	900	450	7170	60
Coach	66	33	1078	62

How to run the multibody code (homework?)

```
1. Log into the VM.
2. cd ~/ADIC/
3. source setenv_inst.sh            -- Sets required environment variables
4. cd examples/multibody/Sparsity/adic_output
                                    -- Go into the example directory
5. make clean
6. make multiplefourbar1_20         -- Compile generated files
7. ./multiplefourbar1_20 multipleFourBar1-20 ../bin/multipleFourBar1-20/
                                    -- Run the code and generate
                                       seed_matrix.dat
                                       sparsity_pattern.dat
                                       vertex_colors.dat
8. cd examples/multibody/SplitFiles_Seed/adic_output
                                    -- Go into the example directory
                                    -- You will be using prior generated files.
9. make clean
10. make multiplefourbar1_20         -- Compile generated files
11. ./multiplefourbar1_20 multipleFourBar1-20 ../bin/multipleFourBar1-20/
```